CGM, Graphcap, and Fontcap Supplement

Author: Fred Clare

Technical writer/editor: Brian Bevirt

Table of Contents

NCAR Computer Graphics Metafile translator

Background

A "metafile" or a "graphics metafile" is a file of encoded graphics instructions. Execution of applications programs that access the current NCAR Graphics utilities produce metafiles. For example, execution of the GKS POLYLINE instruction for drawing a line segment will result in the encoding of that command and the appropriate coordinate positions, and the placing of this encoding into the output metafile. The encoded commands in most metafiles are "device-independent" in that they are not specific to any particular graphics device. On the other hand, each graphics output device, such as a graphics terminal like the Tektronix 4107, or a film recorder, such as the Dicomeds at NCAR, is driven by a special command set associated with that given device. A metafile translator is a program module that will decode the generic device-independent instructions in a metafile and convert them to the device-specific commands for driving a given graphics output device. Thus we speak of the t4107 translator, or the Dicomed translator.

Metafiles have two primary values in a complex computing environment: 1.) simplification, and 2.) archivability and transportability.

When a computing environment contains a variety of computers and graphics peripherals, it is faced with the problem of providing a uniform graphics package running on m different hosts and driving n different output devices. The basic desire is to create graphics on any of the hosts, and plot the results on any of the output devices. One way to proceed would be to provide a plotting application for each host/device pair. Another way to proceed is to create a generic metafile on the host (the same application run on different hosts will produce identical metafiles), and then invoke a metafile translator to drive a given output device (at most one translator for each device). Thus, the metafile step reduces the complexity from producing m x n separate packages to producing one portable application running on all hosts and a metafile translator for each output device.

In the current NCAR software, simplification has been taken one step further with the advent of a table-driven translator. For any output device which has user access to its native command set, it is necessary only to complete filling in a device description table (called a "graphcap" file); the master translator will configure itself appropriately for a given output device, given the graphcap file for that device. Providing a graphcap for a particular device may not be trivial in general, but it is much easier than writing a special translator for a given device.

Another value of metafiles is for archivability and transportability. If one has a set of plots that one wants to store for later examination, the plots can be captured in a metafile. Also, if one wants to create plots at one site, and then view them somewhere else, the plots can be transported via a metafile. Many NCAR scientists and visitors use this feature, which is of course dependent on their having access to a translator at their remote site. A variant on the transportability of metafiles is their use as a picture storage medium in a batch environment where the host has no direct link to drive the output device.

There are two primary drawbacks with having a metafile step between the picture generation and viewing. One drawback is overhead---given a host/plotting device pair, it would be more efficient if the host generated the device instructions for the given device directly and displayed the result. In exchange for simplicity and flexibility, we have compromised efficiency. Also, metafiles are limited in interactive work---for example, there is no support for graphical input. If you don’t need to create a metafile to permanently store your graphical output, then you may want to use the X11 workstation of NCAR Graphics. Please refer to the NCAR Graphics Fundamentals for more information.

The metafile standard

The concept of "metacode," meaning a device-independent graphic picture encoding, dates from the early 1970s. Over the years "metacode" has come to be internationally known as "metafile."

As the concept of metafile gained importance in the international graphics community, it became clear that national and international standards were desirable. One of the main uses of metafiles is to transport picture encodings from one site to another, and if everybody has their own definition for a metafile, going from one site to another requires transformation from one metafile format to another. Complete transformation from one metafile format to another can be difficult or impossible, since the functionality of one format may be considerably different from that of another. Certainly, the inconvenience and possible introduction of errors make non-standard metafile formats undesirable.

In 1986, ANSI (the American National Standards Institute) accepted the CGM (Computer Graphics Metafile) as a national standard. The ANSI committee that produced the CGM contained representatives from several dozen organizations, including almost all of the major computing vendors and government labs. The CGM has also been accepted as an ISO (International Standards Organization) standard.

The CGM is a standard that defines a set of basic elements for a computer graphics interface. The CGM is what is known as a "picture capture" metafile. The conceptual structure of the metafile is one of a sequence of distinct self-contained pictures. This is to be contrasted with an "audit trail" metafile (such as the one described in Annex E of the GKS standard), which is a transcription of a graphics session. Pictures can be extracted at random from a CGM and plotted, whereas to plot a picture from a GKS metafile, one has to sequentially process all instructions in the metafile prior to the desired picture before plotting the desired picture. NCAR has decided to adopt the CGM as its standard metafile. This will replace the earlier pre-CGM NCAR metafile definition which dates to the early 1970s. The CGM specifies three alternative encoding schemes for the defined elements. The encoding scheme that has been adopted at NCAR is the binary encoding. Of the three possible encoding schemes, the binary encoding is the most machine efficient. The physical record structure for a CGM file is not specified in the CGM standard. NCAR has specified a record structure for the NCAR CGM that is similar to the record structure of the pre-CGM NCAR metafiles; it is the record structure only that makes NCAR's CGM definition specific to NCAR.

It should be emphasized that the CGM is a distinct and separate standard from the GKS standard, which is an ANSI and ISO standard specifying a set of basic functions for computer graphics programming. The CGM standard is not totally unrelated to the GKS standard, however. The CGM elements capture much of the functionality of the GKS functions, but far from all of them---for example, the picture segmentation of GKS is absent from the current set of CGM elements. Additionally, the CGM contains many elements that are not a part of the GKS standard---for example, the CIRCULAR ARC elements.

The CGM offers two significant advantages over the old NCAR metafile; it standardizes the instruction set, and it supports several dozen commands instead of just a few. The CGM supports raster interfaces, polylines, bundled attributes, and filled areas.

There are over 90 elements in the CGM standard; the NCAR CGM does not support all of these elements. For a detailed description of the NCAR CGM, see the NCAR Computer Graphics Metafile section of this document.

The CGM standard itself is contained in the document:

ANSI X3.122-1986
Information Processing Systems
Computer Graphics

Metafile for the Storage and Transfer of Picture Description Information

This document is available from:

ANSI
1430 Broadway
New York, NY 10018
Phone: (212) 354-3300

Subsequent to NCAR's implementing the CGM Standard, significant additions have been made to the Standard. These were officially added in 1992.

The 1992 version of CGM (referred to as CGM:1992) is ISO Standard 8632 and has four parts (the prices may be out of date):

ISO 8632-1 - Functional Specification (ANSI price $210)
ISO 8632-2 - Character Encoding (ANSI price $123)
ISO 8632-3 - Binary Encoding (ANSI price $119)
ISO 8632-4 - Clear Text Encoding (ANSI price $101)

There have also been amendments published in 1994:

ISO 8632-1 - Amendment 1. (ANSI price $145)
ISO 8632-2 - Amendment 2. (ANSI price $37)
ISO 8632-3 - Amendment 3. (ANSI price $37)
ISO 8632-4 - Amendment 4. (ANSI price $34)

All ANSI documents are copyrighted, so there is no legal online free source.

There is a book: *The CGM Handbook* by L.R. Henderson and A.M. Mumford, published by Academic Press in 1993.

This book is very complete and thorough and is highly recommended. The book covers the most recent version of the CGM (ISO/IEC 8632) published in 1992.

Graphcap files

A graphcap file is a table describing the characteristics of a graphics device. This section explains how to create a graphcap for a graphics output device. When the table is complete, it must be input into the graphcap preprocessor graphc to produce a binary version of the graphcap, which is in turn used by an NCAR CGM translator at execution time. Details on how graphcaps fit into the overall picture of metafile translation are provided in the introductory section of this document, NCAR Computer Graphics Metafile Translator.

This "Graphcap files" section is oriented toward the user who is familiar with the operation and instruction set of the target computer graphics display device. Casual users will find that generating a graphcap from scratch is not an easy task. However, it is relatively easy to modify an existing graphcap to produce a different color table, change the frame finished prompt, modify the picture orientation, or select a different line type. This allows users to customize the translator to their own requirements without modifying program code.

Graphcaps are used to define device-specific commands. If the interface to a graphics output device is via subroutine calls, then one cannot utilize the graphcap concept. Instructions for implementing the C translator for devices with a software interface are not included in this document, but may appear in future editions. The ambitious installer may want to make the necessary changes to the translator code itself in order to implement the C translator on a device with a software interface.

In its original form, the graphcap consists of a set of keywords followed by the definition of that operation for a selected graphics device. Only those keywords needed for a given graphcap need appear in that graphcap. Usually, if the device does not support the option, the keyword is left out of the graphcap. If a keyword is present but not defined, it is equivalent to the keyword's not being present. In both of these cases, the default definition (if any) will be used. A keyword must be contained on one line; typically a keyword will appear on one line, and the definitions for that keyword will appear on subsequent lines. Complete descriptions of all graphcap keywords appear later in this section.

To create or modify a graphcap, use any text editor that does not insert any of its own control characters into the file.

If you are creating a graphcap from scratch, you first need to find out from your site installer (or the "NCAR Graphics UNIX Installer's Guide") where you can get the NCAR Graphics source. Then you can use the master form of a graphcap which is in a file called **form.gc** in the directory **$NCARG/common/src/graphcap**. This master file contains keywords for all currently supported graphcap functions.

There are four types of keyword definitions:

1. Logical, valued TRUE or FALSE. These values are stored, for the target machine, in a Fortran type LOGICAL variable. Note that the values specified here are TRUE and FALSE, not ".TRUE." and ".FALSE.".
2. Decimal valued. Positive or negative decimal values are specified, separated by blanks. These values are stored, for the target machine, in a Fortran type INTEGER.
3. Floating point valued. A floating point value must contain a decimal point. Floating point values are separated by blanks. The value is stored, for the target machine, in a Fortran type REAL.
4. String type. A string consists of ASCII characters plus special control values that control the placement of addresses, coordinates, and counts within instruction strings. Each character in the keyword string definition is separated by one or more blanks or a new line. The three-character sequences in the initial part of the ASCII table are considered as one character in a string definition. The ASCII decimal equivalent for each character in the string is stored, for the target machine, in the lower eight bits of a Fortran type INTEGER. There are seven special characters recognized (these are actually character sequences, but are treated as a single character in keyword string definitions) by the translator. These special characters are:

- INT*nnn* where *nnn* can be any integer between 0 and 255.
  This character is used when one wants to insert an actual integer value into the keyword string definition.
- MAD This character is used to indicate that a color map address is to be inserted into the keyword

string definition.

- RL This character is used to indicate that run-length encoding will be used to generate raster instructions.
- VC This character is used to indicate that a vector count is to be inserted into the keyword string definition.
- XC This character is used to indicate that an X-coordinate is to be inserted into the keyword string definition.
- YC This character is used to indicate that a Y-coordinate is to be inserted into the keyword string definition.
- XYC This character is used to indicate that an X/Y-coordinate pair is to be inserted into the keyword string definition.

To better understand the use of these special characters, examine their use in some of the graphcaps supplied on the distribution tape.

All keyword definitions may range from null to a maximum count specified by that keyword. Keyword definitions are allowed a maximum of 80 columns per line, but they may cross line boundaries. For separating keyword definition entries, a new line is equivalent to a space.

To understand and change the bundle class, refer to the GKS standard, in particular the description there of the workstation state list. The bundle class must be included and defined in every graphcap used. The blank graphcap form defines the bundle entries as required by GKS. A user may make nonstandard definitions by changing these entries. Comment any changes made to bundle entries, this will alert others to non-standard features in the graphcap.

When defining a new device, it may be easiest to start with an existing graphcap that is similar, and work from there. For example, many of the Tektronix emulators only require changes to the DEVICE_GRAPHIC_INIT and DEVICE_TEXT_INIT keyword definitions to operate properly.

As mentioned earlier, a blank form is included with the graphcap files; use this as a starting point for generating a new device table.

Keywords may be in any order, the groupings in subsequent sections are for description only.

When reading the next section, it is best to get a copy of a graphcap for a device you are familiar with and use it as an example of the keyword definitions.

Please see Appendix A for a list of all the available graphcaps in Version 4.*x* of NCAR Graphics.

Formatting and encoding coordinates

The process of creating coordinates, color/intensities, and other parameters that are part of a device instruction is normally not simple. Many devices require that bits from various parts of the data values be packed together to form the device instruction parameters. For example, the Tektronix 40xx series devices require that the first word of a device coordinate parameter contain five bits from the upper part of the Y coordinate packed with some flagging information. Other devices require that parts of the X and Y coordinate be packed together. One describes this formatting information to the translator by filling in a format table that contains the bit positions for extracting from input data and inserting into device parameters.

Once the data are successfully packed, many devices require a second massaging prior to sending the packed data to the output device. Encodings take the bit stream created by the formatting process and convert to another bit stream, which is sent to the device. For example, the Tektronix 41xx terminal series requires that the bit stream from the formatting process be converted into a set of printable characters.

Many devices require that parameters be converted in these two steps. The present discussion explains the conversion process used for all keywords that contain "FORMAT" or "ENCODING" as a substring in the keyword.

The formatting process

The formatting process maps an input array of integers into an output stream of bits.

The words in the input array are addressed as in Fortran---that is, the first element of the array is at position 1. The bits of the input words are addressed from right to left. The first bit of each element is at position 0 (this is the rightmost bit, values increase to the left).

The output stream is a string of bits; there is no regard for word boundaries. The stream is addressed from left to right. The first bit is at address 0.

Data are extracted from input words and inserted into the output stream according to the information given in a formatting table as described below. The table contains four entries per row. Each row defines a separate operation for extracting from an input word and inserting into the output stream. Each row contains a start_bit, bit_count, data_type, and data_value. All four entries must be defined for each row. All four entries in the row are integer values. Since several graphcaps depend on the word size of the host machine, it is legal to use one of two special symbols to denote integers in the format tables. These symbols are "W" to denote "the word size, in bits, of the host machine" and "W1" to denote W-1 (the word size minus 1).

1. start_bit The starting index in the output string.
2. bit_count The number of bits to be transferred.
3. data_type This determines where input data originates for the extraction step. Valid values for data_type are:

   [-1]

   This data_type indicates that bits will be extracted from the data_value field and inserted into the output stream. The output stream is then sent to the encoder, which has the effect of clearing the stream (the output stream is disposed to the encoder and then to the device). If there is another row in the format table to follow, addressing of the output stream should begin from the left edge (address 0). The -1 data_type is typically used with the ASCII encodings to signal the end of a number to be encoded and to insert a separator into the device stream. The bit_count must be divisible by 8 (to fill 8 bit bytes).

   [0]

   This value indicates that the contents of the data_value field should be moved intact to the output stream.

   [>0]

Use the addressed input word, e.g. if the value is 2, the second input word is used as input.

4. data_value The use of data_value is controlled by the type of operation requested by data_type. For data_types of -1 or 0, the data_value is used as the input word; for data_types > 0, the data_value indicates the bit position in the addressed input word (position 0 at the right edge of the word) where data transfer is to begin. For example, if data_type is 2, bit_count is 5, and data_value is 10, then 5 bits from word 2 would be put to the output stream starting at bit position 10 in word 2 and continuing for 5 bits to the right in word 2.

In most cases you will not know how many input words will be received. So how can you enter enough data values greater than zero to cover the input stream? If you request too many data values, unexpected and unpredictable things will happen. However, you do not need to specify enough entries for all input data. The format table describes one coordinate pair and will cycle until all input data have been processed. Suppose you have a table defining X/Y coordinates and the Xs and Ys are packed together in some strange way (Tektronix 4010 for example). You just define how the first two words are packed (word 1 is the X and word 2 is the Y). The table will cycle until all the input words have been processed. For a color table you may have three entries for each color definition, so define how those three entries are constructed and the table will cycle for all the rest of the data.

The following example for the Tektronix 4010 coordinate formatting demonstrates how to construct a graphcap format table. Also, in the following sections of this document, all keywords related to formatting contain examples specific to their subject. In the following chart, each line describes an 8-bit byte, which is to be sent to the device. The letter P stands for parity, and the letters X and Y stand for the X and Y coordinates. The digit following X or Y is the bit being referred to in the coordinate as described for the input data indexing above. The 0s and 1s are required by those bit positions. This description is taken from a Tektronix Terminal Programmer's Manual.

```
---------------------------
P  0  1  Y9  Y8  Y7  Y6  Y5
P  1  1  Y4  Y3  Y2  Y1  Y0
P  0  1  X9  X8  X7  X6  X5
P  1  0  X4  X3  X2  X1  X0
---------------------------
```

The graphcap table to describe this encoding is:

```
----------------------------------------------------------------------
DEVICE_COORD_FORMAT
/*  bit_start  bit_count  data_type  data_value
       1           2          0          1
       3           5          2          9
       9           2          0          3
      11           5          2          4
      17           2          0          1
      19           5          1          9
      25           2          0          2
      27           5          1          4
----------------------------------------------------------------------
```

More examples are provided in each FORMAT type keyword.

The encoding process

The encoding process takes the output stream from the formatter and performs another massaging of the data prior to disposing to the device. The encoding is defined by one of the decimal values described below.

0 - binary encoding
: No change to the formatted data.

1 - ASCII decimal encoding
: The bit stream from the formatter is translated into an ASCII string representing the decimal equivalent of the string. The bits 1111 would be converted to the ASCII sequence 15 and sent to the device as an integer 49 and an integer 53.

2 - ASCII hexadecimal encoding
: The bit stream from the formatter is translated into a ASCII string representing the hexadecimal equivalent of the string. The bits 1111 would be converted to an ASCII F and sent to the device as an integer 170 (the ASCII decimal equivalent of F).

3 - ASCII octal encoding
: The bit string from the formatter is translated into a ASCII string representing the octal equivalent of the string. The bits 1000 (octal 10) would be converted to two ASCII characters 1 0 and sent to the device as integer 49 (the ASCII decimal equivalent of 1) and integer 48 (the ASCII decimal equivalent of 0).

4 - ASCII Tektronix encoding
: This type of encoding is used by the Tektronix 41xx class of devices.

5 - ASCII Real
: This encoding translates a bit stream into a printable floating point number. The conversion from the bit stream, which is interperted as a whole number, depends on the corresponding FLOATING_INFO keyword. The FLOATING_INFO type keywords provide four real numbers to the encoder. The first two are the minimum and maximum values expected from the bit stream. The last two specify the minimum and maximum floating point values to generate. A linear mapping is performed from input to output.

Workstation initializations

The following keywords (presented in boldface) describe the strings used to set the device into graphics or text mode. Graphics mode is the state where graphics instructions are received and understood; text mode is used for normal communications and editing.

DEVICE_GRAPHIC_INIT

Description: Enter graphics mode. Sent at invocation of the translator, and at the beginning of each new frame when the DEVICE_BATCH keyword (see below) is set to FALSE.

Type: String

Default: Null

Maximum number of entries: 300 characters

DEVICE_TEXT_INIT

Description: Enter text (non-graphics) mode. Sent at termination of the translator.

Type: String

Default: Null

Maximum number of entries: 100 characters

DEVICE_BATCH

Description: Logical value indicating whether the translator sends the USER_PROMPT and waits for keyboard carriage return. This operation happens at completion of a picture. Batch devices may include laser printers and film devices. Non-batch devices may be graphics display terminals. Batch devices do not prompt for user intervention, they automatically proceed to the next frame.

Type: Logical

Default: FALSE, user intervention is required to advance pictures

Maximum number of entries: 1

DEVICE_ERASE

Description: String sent to the device when the display surface is to be cleared.

Type: String

Default: Null

Maximum number of entries: 150 characters

DEVICE_CURSOR_HOME

Description: String sent to the device when the cursor is to be positioned at the home position. The translator considers home as the upper left corner of the picture-drawing surface.

Type: String

Default: Null

Maximum number of entries: 50 characters

USER_PROMPT

Description: The string sent to the device indicating to the user that the picture on the display surface is finished. At this point, some device dependent action (usually a carriage return) is required to continue. This string is used only if DEVICE_BATCH is set FALSE. The prompt is typically displayed in graphics mode. On small single-page plotters you may want to terminate graphics mode in this string and then send the prompt. This has the effect of sending the prompt to the CRT connected to the plotter but not displaying it on the plotter.

Type: String

Default: Null

Maximum number of entries: 80 characters

Workstation drawing space

This set of keywords describes the drawing surface available on the workstation. The picture can be drawn in any orientation or transposition by properly changing the lower left and upper right coordinates and perhaps interchanging the order of coordinate processing as defined by the DEVICE_COORD_FORMAT.

For example, if your device has lower left X, Y at (0, 0) and upper right at (1000, 1000), you could draw the picture upside down by setting the lower left coordinates to (0, 1000) and the upper right coordinates to (1000, 0). The GKS standard (section 4.6.3) requires that the aspect ratio of the workstation viewport be the same as that of the workstation window. To implement this, the translator examines the device coordinate ranges as specified in the following keywords and chooses the largest square that can be displayed; it then centers this square on the output device and uses it as the workstation viewport (unless coordinate offsets are defined using the appropriate keywords below).

DEVICE_COORD_LOWER_LEFT_X

Description: The device X coordinate of the lower left corner of the drawing surface.

Type: Decimal

Default: Null

Maximum number of entries: 1

DEVICE_COORD_LOWER_LEFT_Y

Description: The device Y coordinate of the lower left corner of the drawing surface.

Type: Decimal

Default: Null

Maximum number of entries: 1

DEVICE_COORD_UPPER_RIGHT_X

Description: The device X coordinate of the upper right corner of the drawing surface.

Type: Decimal

Default: Null

Maximum number of entries: 1

DEVICE_COORD_UPPER_RIGHT_Y

Description: The device Y coordinate of the upper right corner of the drawing surface.

Type: Decimal

Default: Null

Maximum number of entries: 1

DEVICE_COORD_XOFFSET

Description: A device coordinate value added to X addresses before they are sent to the device. The translator attempts to center the plot based on the lower left, upper right values. When the coordinate space is asymmetric (X coordinate interval is not the same as the Y coordinate interval), offset values will help properly place the frame. Another use is to offset the frame to a specific side of the display.

Type: Decimal

Default: 0 (zero)

Maximum number of entries: 1

DEVICE_COORD_YOFFSET

Description: A device coordinate value added to Y addresses before they are sent to the device. See DEVICE_COORD_XOFFSET description.

Type: Decimal

Default: 0 (zero)

Maximum number of entries: 1

DEVICE_COORD_XSCALE

Description: In some cases, an X unit does not equal a Y unit. This keyword allows a scale factor to be introduced into the device coordinate computation stream. It will scale the X coordinates up or down by the specified amount.

Type: Floating point

Default: 1.0

Maximum number of entries: 1

DEVICE_COORD_YSCALE

Description: See the DEVICE_COORD_XSCALE description.

Type: Floating point

Default: 1.0

Maximum number of entries: 1

DEVICE_COORD_FORMAT

Description: The format used to convert device coordinates in the metafile to the format required by the output device. The input array to this format is a set of coordinate pairs. See the discussion of formatting in the Formatting and encoding coordinates section of this document. Two examples follow. They describe the required bit positions as P for parity, Xn for an X bit and Yn for a Y bit. The parity bit is not relevant to the encoding process, so those bits are skipped over in these examples. We use the AED512 7-bit binary encoding for the first example.

```
-------------------------------
P  X9  X8  X7  Y10  Y9  Y8  Y7
P  X6  X5  X4  X3   X2  X1  X0
P  Y6  Y5  Y4  Y3   Y2  Y1  Y0
-------------------------------
```

The format table will appear as below:

```
--------------------------------------------------------------------
DEVICE_COORD_FORMAT
/*  bit_start  bit_count  data_type  data_value
       1           3          1          9
       4           4          2         10
       9           7          1          6
      17           7          2          6
--------------------------------------------------------------------
```

For the next example, we use the AED512 ASCII decimal encoding. This encoding uses a 16-bit address, and each coordinate is terminated by a blank. The X coordinate is first, followed by the Y coordinate.

```
--------------------------------------------------------------------
DEVICE_COORD_FORMAT
/*  bit_start  bit_count  data_type  data_value
      0           16          1          15
      0            8         -1          32
      0           16          2          15
```

```
      0               8           -1             32
--------------------------------------------------------------------
```

Type: Decimal rows with four elements per row

Default: Null

Maximum number of entries: 120 (30 rows * 4 columns)

DEVICE_COORD_ENCODING

Description: The encoding scheme used for device coordinates. See the Formatting and encoding coordinates section of this document.

Type: Decimal in the range of 0 to 5

Default: 0 (zero)

Maximum number of entries: 1

DEVICE_COORD_FLOATING_INFO

Description: The mapping from fixed point (integer) to output floating point. This option is used when the DEVICE_COORD_ENCODING is set to ascii real (5). The keyword requires four floating point numbers. The first two describe the minimum and maximum data values received by the encoder. The last two describe the minimum and maximum floating point values sent to the device.

Type: Floating point

Default: Null

Maximum number of entries: 4

Device vector counts

The vector count is a parameter that indicates the number of points (X/Y coordinate pairs) that are part of the current instruction. These keyword definitions are activated by the VC parameter, which may be included in the LINE, MARKER, and POLYGON instruction start keywords. VC may also be used in the RASTER_HORIZONTAL instruction start keyword.

DEVICE_VECTOR_COUNT_ENCODING

Description: The encoding scheme used for vector counts. See the section Formatting and encoding coordinates.

Type: Decimal in the range of 0 to 5

Default: 0 (zero)

Maximum number of entries: 1

DEVICE_VECTOR_COUNT_FLOATING_INFO

Description: The mapping from fixed point (integer) to output format floating point. This option is used when the DEVICE_VECTOR_COUNT_ENCODING is set to ascii real (5). The keyword requires four floating point numbers. The first two describe the minimum and maximum data values received by the encoder. The last two describe the minimum and maximum floating point values sent to the device.

Type: Floating point

Default: Null

Maximum number of entries: 4

DEVICE_VECTOR_COUNT_FORMAT

Description: The format used to convert vector count data from the CGM to the format required by the output device. This formatter will receive only one input data word. See the section Formatting and encoding coordinates.

The following example encodes a 7-bit vector count. The index is right-justified in the vector count parameter passed into the formatter. The P represents parity bit, and V followed by a digit is a bit out of the vector count. Parity is not relevant and is skipped over. We are using the binary encoding scheme.

```
-----------------------------
P  V6  V5  V4  V3  V2  V1  V0
-----------------------------
```

This format would be set up as follows:

```
---------------------------------------------------------------------------
DEVICE_VECTOR_COUNT_FORMAT
/*  bit_start  bit_count  data_type  data_value
        1          7          1          6
---------------------------------------------------------------------------
```

Type: Decimal rows with four elements per row

Default: Null

Maximum number of entries: 32 (8 rows * 4 columns)

Device color capabilities

This section describes the general color availability on a given device. For specific color operations, refer to the LINE, MARKER, and POLYGON primitives referenced in later parts of this section.

DEVICE_COLOR_AVAILABLE

Description: Flag to indicate the availability of color on a device. If the device color is TRUE then the DEVICE_MAP_INDEX_RANGE_DEFINED keyword must be defined in the graphcap. If the device has multiple monochrome intensities, then this keyword should be defined as TRUE. If DEVICE_MAP_AVAILABLE is TRUE (see below) and DEVICE_MAP_MODEL is 0 (monochrome---see below), then RGB triples are mapped to monochrome intensities by the formula Y = 0.3*R + 0.6*G + 0.1*B as per the CGM Standard (p. 126).

Type: Logical

Default: FALSE, no color on the device

Maximum number of entries: 1

DEVICE_COLOR_INDEX_ENCODING

Description: The encoding scheme used for color indices. See the section Formatting and encoding coordinates.

Type: Decimal in the range of 0 to 5

Default: 0 (zero)

Maximum number of entries: 1

DEVICE_COLOR_INDEX_FORMAT

Description: The format used to convert color indices in the CGM to the format required by the output device. The formatter will receive only one input data word. See the section Formatting and encoding coordinates.

The following example uses a 4-bit color index (0 to 15 decimal). The P represents parity bit; 0s (zeros) and 1s represent themselves and c followed by a digit is a bit out of the color index. Parity is not relevant and is skipped over. We are using the binary encoding scheme.

```
--------------------------
P  0  1  1  c3  c2  c1  c0
--------------------------
```

This format would be set up as follows:

```
---------------------------------------------------------------------------
DEVICE_COLOR_INDEX_FORMAT
/*  bit_start  bit_count  data_type  data_value
       1          3          0          3
       4          4          1          3
---------------------------------------------------------------------------
```

Type: Decimal rows with four elements per row

Default: Null

Maximum number of entries: 60 (15 rows * 4 columns)

DEVICE_COLOR_INDEX_FLOATING_INFO

Description: The mapping from fixed point (integer) to output format floating point. This option is used when the DEVICE_COLOR_INDEX_ENCODING is set to ascii real (5). The keyword requires four floating point numbers. The first two describe the minimum and maximum data values received by the encoder. The last two describe the minimum and maximum floating point values sent to the device.

Type: Floating point

Default: Null

Maximum number of entries: 4

DEVICE_MAP_AVAILABLE

Description: A flag indicating the availability of a user-settable color/intensity map.

Type: Logical

Default: FALSE, no user-settable map available

Maximum number of entries: 1

DEVICE_MAP_INDEX_RANGE_MAX

Description: The size of the user-settable color/intensity map. If the map has 256 entries (0 to 255), then use 256.

Type: Decimal

Default: 0 (zero)

Maximum number of entries: 1

DEVICE_MAP_INDEX_RANGE_DEFINED

Description: The range of colors defined by the default settings. If the device has entries 0 to 7 defined, then use 8. This keyword must be defined if the device has color capability, even if the color map is not settable.

Type: Decimal

Default: 0 (zero)

Maximum number of entries: 1

DEVICE_MAP_INTENSITY_ENCODING

Description: The encoding scheme used for intensity settings. See the section Formatting and encoding coordinates.

Type: Decimal in the range of 0 to 5

Default: 0 (zero)

Maximum number of entries: 1

DEVICE_MAP_INTENSITY_FLOATING_INFO

Description: The mapping from fixed point (integer) to output format floating point. This option is used when the DEVICE_MAP_INTENSITY_ENCODING is set to ascii real (5). The keyword requires four floating point numbers. The first two describe the minimum and maximum data values received by the encoder. The last two describe the minimum and maximum floating point values sent to the device.

Type: Floating point

Default: Null

Maximum number of entries: 4

DEVICE_MAP_INTENSITY_FORMAT

Description: The format used to convert the color/intensity values in the CGM to the format required by the output device. This formatter expects sets of three input data words if in a full-color mode and single input data words if in a monochrome mode. Observe the setting of the DEVICE_MAP_MODEL keyword for the mode used by this formatter. See the section Formatting and encoding coordinates.

For the following example, assume an RGB color model with 100 map entries. The formatter will have 300 words passed to it (three per entry). The device uses eight bits per intensity. We are using a binary encoding for this example.

```
----------------------------------------------------------------------------
DEVICE_MAP_INTENSITY_FORMAT
/*  bit_start  bit_count  data_type  data_value
        0          8          1          7
----------------------------------------------------------------------------
```

All the intensity settings are treated the same, so we only need one entry in the table to format all 300 settings.

The next example also has 100 RGB map entries. The device uses an ASCII decimal encoding. Red and green are separated by a blank. Green and blue are separated by a period. Green and the red of the next entry are separated by a colon.

```
-----------------------------------------------------------------------------
DEVICE_MAP_INTENSITY_FORMAT
/*  bit_start  bit_count  data_type   data_value
       0          8          1           7
       0          8         -1          32
       0          8          2           7
       0          8         -1          46
       0          8          3           7
       0          8         -1          58
-----------------------------------------------------------------------------
```

Type: Decimal rows with four entries per row

Default: Null

Maximum number of entries: 200 (50 rows * 4 entries per row)

DEVICE_MAP_INDIVIDUAL

Description: A flag that indicates a DEVICE_MAP_INSTRUCTION_START and a DEVICE_MAP_INSTRUCTION_TERMINATOR needs to be sent for each entry. If the flag indicates no need, the DEVICE_MAP_INSTRUCTION_START and DEVICE_MAP_INSTRUCTION_TERMINATOR are sent only once for map intensity streaming.

Type: Logical

Default: FALSE, DEVICE_MAP_INSTRUCTION_START and DEVICE_MAP_INSTRUCTION_TERMINATOR are sent only once for map intensity streaming

Maximum number of entries: 1

DEVICE_MAP_INSTRUCTION_START

Description: The string sent to the device to enable it to receive a new intensity map definition. The MAD string parameter is used in this string if map addresses are required by the device instruction.

Type: String

Default: Null

Maximum number of entries: 50

DEVICE_MAP_INSTRUCTION_TERMINATOR

Description: Terminate the color map instruction.

Type: String

Default: Null

Maximum number of entries: 20 characters

DEVICE_MAP_INIT

Description: The intensity map used for initialization of the device prior to display of each picture. These data are relative to the device model selected via the DEVICE_MAP_MODEL and are given in device-dependent intensity values.

When the encoding is ASCII real (value 5), then the values given here must be integers in the range of the first two values of the DEVICE_MAP_FLOATING_INFO. The map intensities are mapped to the floating values when sent to the encoding process. This is necessary because this keyword allows only decimal input.

Type: Decimal

Default: Null

Maximum number of entries: 768 (256 * 3)

DEVICE_MAP_MODEL

Description: This keyword defines the type of intensity model used by the device. There are four types included in the translator.

0 - Monochrome
This model is used on monochrome devices (one color) that have variable intensities. A good example is a black and white film device such as the Dicomed COM devices.
1 - RGB
Red, green, blue, a popular model on color devices. Intensities are given as triples.
2 - BGR
Blue, Green, red. Not common, but used on some Ramtek color devices.
3 - HLS
The Hue, Lightness, and Saturation model, which is easier for most users to work with than RGB. This model is used by Tektronix color devices.
Type: Decimal in the range of 0 (zero) to 3

Default: 1 (RGB)

Maximum number of entries: 1

Device window

This section describes keywords for specifying a workstation window (in the GKS sense). Four coordinates are specified to define a rectangular window which is a subset of the normalized VDC rectangle with corner points (0, 0) and (32767, 32767). The specified window is then mapped onto the entire viewport. For example, if the workstation window is defined by the corner points (0, 0) and (16383, 16383) then the lower left quarter of a plot would be expanded to fill the entire viewport. Specification of such a window can be used for zooming and panning.

DEVICE_WINDOW_LOWER_LEFT_X

Description: The X coordinate of the lower left corner of the window in the range 0 to 32767.

Type: Decimal

Default: 0

Maximum number of entries: 1

DEVICE_WINDOW_LOWER_LEFT_Y

Description: The Y coordinate of the lower left corner of the window in the range 0 to 32767.

Type: Decimal

Default: 0

Maximum number of entries: 1

DEVICE_WINDOW_UPPER_RIGHT_X

Description: The X coordinate of the upper right corner of the window in the range 0 to 32767.

Type: Decimal

Default: 32767

Maximum number of entries: 1

DEVICE_WINDOW_UPPER_RIGHT_Y

Description: The Y coordinate of the upper right corner of the window in the range 0 to 32767.

Type: Decimal

Default: 32767

Maximum number of entries: 1

Line control

Used for drawing vectors and stroked characters. This set of instructions may also be used by the translator to position the device graphics cursor.

LINE_DRAW_POLY_FLAG

Description: Some devices support polylines, which are distinguished by the polyline instruction start, followed by the coordinates for the entire line. Other devices do not support polyline, which requires that each coordinate pair be accompanied by an instruction, whether it is a move or draw.

This flag indicates the availability, or lack of availability, of a polyline instruction on the device.

When a polyline instruction is available, the LINE_DRAW_INSTRUCTION_START initiates the polyline and the LINE_DRAW_INSTRUCTION_TERMINATOR ends the polyline.

When there is no polyline on the device, the MOVE set is used for moving and the DRAW set is used for drawing.

Type: Logical

Default: FALSE, no polyline available on the device

Maximum number of entries: 1

LINE_DRAW_INSTRUCTION_START

Description: The instruction that produces a line on the display surface. If the device has polylines, then this instruction is sent at the start of a polyline.

Type: String

Default: Null

Maximum number of entries: 30

LINE_DRAW_INSTRUCTION_TERMINATOR

Description: Terminate the draw instruction. This is also used to terminate the polyline instruction.

Type: String

Default: Null

Maximum number of entries: 15

LINE_POINT_START

Description: Specifies the instruction to be issued prior to the encoding of each coordinate pair when LINE_DRAW_POLY_FLAG is TRUE.

Type: String

Default: Null

Maximum number of entries: 20

LINE_POINT_TERMINATOR

Description: Specifies the instruction to be issued subsequent to the encoding of each coordinate pair when LINE_DRAW_POLY_FLAG is TRUE.

Type: String

Default: Null

Maximum number of entries: 20

LINE_MOVE_INSTRUCTION_START

Description: Move the device graphics cursor. This string is not used when the device has polylines.

Type: String

Default: Null

Maximum number of entries: 30

LINE_MOVE_INSTRUCTION_TERMINATOR

Description: Terminate the move instruction. This string is not used if the device has polylines.

Type: String

Default: Null

Maximum number of entries: 15

DASH_BIT_LENGTH

Description: The number of VDC (virtual device coordinate) units used for each bit of a software dash line pattern. This dash pattern is defined in the CGM or by the bundle tables. Modifying this allows all devices to generate a similar-sized dash. The default dash patterns are located in the translator source file BLOCKDATA TRNDAT, which is a BLOCKDATA routine. If you select GKS line indices 1-4, you will get one of these dash patterns.

Type: Decimal

Default: 100; there are 100 VDC units per dash-pattern bit

Maximum number of entries: 1

LINE_COLOR_INSTRUCTION_START

Description: The instruction to change the line color. A color index is sent after this string.

Type: String

Default: Null

Maximum number of entries: 30

LINE_COLOR_INSTRUCTION_TERMINATOR

Description: Terminate the line color instruction.

Type: String

Default: Null

Maximum number of entries: 15

LINE_BACKGROUND_COLOR_INSTRUCTION_START

Description: The instruction to change line color when the background color is being used. This is provided for support of those few devices that require special instructions for drawing in the background color.

Type: String

Default: Null

Maximum number of entries: 30

LINE_BACKGROUND_COLOR_INSTRUCTION_TERMINATOR

Description: Terminate the line color instruction when using the background color.

Type: String

Default: Null

Maximum number of entries: 15

Line widths

This section describes keywords for controlling line widths. First we give a brief description of how the Fortran translator handles line width. The translator first checks to see if there is a line width command available on the device (the graphcap entry LINE_WIDTH_INSTRUCTION_START is not empty). If there is no line width instruction on the device, the LINE WIDTH elements are ignored, hence all lines are drawn using the nominal line width on the device. The Fortran translator only supports the default value for LINE WIDTH SPECIFICATION MODE, which is "1" for "scaled," so all LINE WIDTH elements are interpreted as line width scale factors. If the device does have a line width instruction, then a line width is computed in device coordinates. The equation used is:

```
DCWDTH = new_scale_value * 8. * LWTSCF
```

where DCWDTH is type INTEGER and all other values are floating point. In this equation, "new_scale_value" is the floating-point value received from the CGM LINE WIDTH element, and LWTSCF is the value given by the LINE_WIDTH_SCALE graphcap entry (1.0 by default). Now, after computing DCWDTH, the computed value is compared with the values provided by the LINE_WIDTH_RANGE graphcap entry. If the computed value is less than the smaller range value, then DCWDTH is set to the smaller range value; if the computed value is greater than the larger range value, then DCWDTH is set to the larger range value. In the default case, the interpretation of a LINE WIDTH value of 1.0 would yield a computed device-specific line width of 8.

LINE_WIDTH_INSTRUCTION_START

Description: Change the line width on the display surface.

Type: String

Default: Null

Maximum number of entries: 30

LINE_WIDTH_INSTRUCTION_TERMINATOR

Description: Terminate the line width instruction.

Type: String

Default: Null

Maximum number of entries: 15

LINE_WIDTH_ENCODING

Description: The encoding scheme used for line width values. See the section Formatting and encoding coordinates.

Type: Decimal in the range of 0 to 5

Default: 0 (zero)

Maximum number of entries: 1

LINE_WIDTH_FLOATING_INFO

Description: The mapping from fixed point (integer) to output format floating point. This option is used when the LINE_WIDTH_ENCODING is set to ascii real (5). The keyword requires four floating point numbers. The first two describe the minimum and maximum data values received by the encoder. The last two describe the minimum and maximum floating point values sent to the device.

Type: Floating point

Default: Null

Maximum number of entries: 4

LINE_WIDTH_FORMAT

Description: The format used to convert line widths in the CGM to a format recognized by the output device. The formatter will receive one input data word.

In the following example, a 16-bit binary line width is being sent to the device.

```
-------------------------------------------------------------------
LINE_WIDTH_FORMAT
/*  bit_start  bit_count  data_type   data_value
        0         16          1           15
-------------------------------------------------------------------
```

Type: Decimal rows with four entries per row

Default: Null

Maximum number of entries: 32 (8 rows * 4 entries per row)

LINE_WIDTH_RANGE

Description: Two values giving the smallest and largest widths available in device units. Required if the LINE_WIDTH_INSTRUCTION_START is defined.

Type: Decimal

Default: Null

Maximum number of entries: 2 (always required if this keyword is defined)

LINE_WIDTH_SCALE

Description: A scale factor applied to the default line width setting for the device.

Type: Floating point

Default: 1.0

Maximum number of entries: 1

Marker control

This section controls the generation of polymarkers. The translator only uses software markers, that is, it internally generates the appropriate stroke sequences for the requested markers and then draws them using the line-drawing routines. Since the line instructions are used to draw the markers, the color change instructions below should be the same as those for line color. The translator does the appropriate scaling, depending on the marker size scale factor.

MARKER_DOT_SIZE

Description: This keyword controls the default size of the dot marker (marker number 1) when the markers are stroked in software. This keyword specifies whether "fat" dots should be drawn. A value of 0 means "draw the dot marker at the smallest possible size;" a value of 1 means draw a "fat" dot. Some devices require that a pen move be executed before drawing a line, and on those devices a simple dot will not be produced by doing a pen-up/pen-down sequence to the same coordinate. For such devices it is necessary to draw a small box for a dot (which we call a "fat" dot). Also, many laser printers draw extremely faint dots at the default size, so producing fat dots on such devices is appropriate. Since it takes four line draws to draw a fat dot, drawing time for fat dots is proportionally longer.

Type: Decimal

Default: 0

Maximum number of entries: 1

The graphcap preprocessor also recognizes and parses the four keywords:

> MARKER_VECTOR_COUNT_FORMAT
> MARKER_VECTOR_COUNT_ENCODING
> MARKER_INSTRUCTION_START
> MARKER_INSTRUCTION_TERMINATOR
> The values for these keywords are passed to the translator, but the translator does not respond to them.

The translator does interpret the following keywords:

MARKER_COLOR_INSTRUCTION_START

Description: Change the marker color.

Type: String

Default: Null

Maximum number of entries: 30

MARKER_COLOR_INSTRUCTION_TERMINATOR

Description: Terminate the color instruction.

Type: String

Default: Null

Maximum number of entries: 15

The translator internally generates stroke sequences for all graphical text based on the supplied fontcap file(s). All text attributes (size, scale factor, path, expansion factor, and so on) are applied before the characters are drawn. The translator uses the line-drawing routines to draw the characters.

The translator does not take advantage of hardware characters.

In its current state, the graphcap preprocessor parses the following keywords and their values are passed to the translator, but the translator does not respond to them:

TEXT_INSTRUCTION_START
TEXT_INSTRUCTION_TERMINATOR
TEXT_COLOR_INSTRUCTION_START
TEXT_COLOR_INSTRUCTION_TERMINATOR
TEXT_VECTOR_COUNT_ENCODING
TEXT_VECTOR_COUNT_FORMAT

Bundle tables

These keywords give limited control over device bundle tables. These entries are currently untested and no guarantee is made that they will work. For a detailed description of bundle tables, and what they can be used for, consult the GKS standard. The translator provides for five pre-defined bundle tables for the classes LINE, MARKER, POLYGON, and TEXT. The user may modify these bundle tables, but such modification should be documented for the benefit of the users. The tables are coded in an obvious fashion---the first element of each keyword applies to the first table, and so on. For example, the first element in the keyword BUNDLE_LINE_INDEX is the bundle table index used for the bundle that consists of the first elements of each of the keywords BUNDLE_LINE_TYPE, BUNDLE_LINE_WIDTH, and BUNDLE_LINE_COLOR.

BUNDLE_LINE_INDEX

Description: Define the indices for the LINE bundle tables.

Type: Decimal

Default: 1 2 3 4 5

Maximum number of entries: 5

BUNDLE_LINE_TYPE

Description: Define the line types for the LINE bundle tables.

Type: Decimal

Default: 1 2 3 4 5

Maximum number of entries: 5

BUNDLE_LINE_WIDTH

Description: Define the line width scale factors for the LINE bundle tables.

Type: Floating point

Default: 1. 1. 1. 1. 1.

Maximum number of entries: 5

BUNDLE_LINE_COLOR

Description: Define the line color indices for the LINE bundle tables.

Type: Decimal

Default: 1 2 3 4 5

Maximum number of entries: 5

BUNDLE_MARKER_INDEX

Description: Define the indices for the MARKER bundle tables.

Type: Decimal

Default: 1 2 3 4 5

Maximum number of entries: 5

BUNDLE_MARKER_TYPE

Description: Define the marker types for the MARKER bundle tables.

Type: Decimal

Default: 1 2 3 4 5

Maximum number of entries: 5

BUNDLE_MARKER_SIZE

Description: Define the marker size scale factors for the MARKER bundle tables.

Type: Floating point

Default: 1. 1. 1. 1. 1.

Maximum number of entries: 5

BUNDLE_MARKER_COLOR

Description: Define the marker color indices for the MARKER bundle tables.

Type: Decimal

Default: 1 2 3 4 5

Maximum number of entries: 5

BUNDLE_POLYGON_INDEX

Description: Define the indices for the FILL AREA bundle tables.

Type: Decimal

Default: 1 2 3 4 5

Maximum number of entries: 5

BUNDLE_POLYGON_INTERIOR

Description: Define the fill area interior styles for the FILL AREA bundle tables.

Type: Decimal

Default: 0 0 0 0 0 (hollow)

Maximum number of entries: 5

BUNDLE_POLYGON_STYLE

Description: Define the style indices for the FILL AREA bundle tables.

Type: Decimal

Default: 0 0 0 0 0

Maximum number of entries: 5

BUNDLE_POLYGON_COLOR

Description: Define the color indices for the FILL AREA bundle tables.

Type: Decimal

Default: 1 2 3 4 5

Maximum number of entries: 5

BUNDLE_TEXT_INDEX

Description: Define the indices for the TEXT bundle tables.

Type: Decimal

Default: 1 2 3 4 5

Maximum number of entries: 5

BUNDLE_TEXT_FONT

Description: Define the text fonts for the TEXT bundle tables.

Type: Decimal

Default: 1 1 1 1 1

Maximum number of entries: 5

BUNDLE_TEXT_PRECISION

Description: Define the text precisions for the TEXT bundle tables.

Type: Decimal

Default: 1 1 1 1 1 (character precision)

Maximum number of entries: 5

BUNDLE_TEXT_CEXPN

Description: Define the character expansion factors for the TEXT bundle tables.

Type: Floating point

Default: 1.0 0.6 0.8 1.2 1.4

Maximum number of entries: 5

BUNDLE_TEXT_CSPACE

Description: Define the character spacings for the TEXT bundle tables.

Type: Floating point

Default: 0. 0. 0. 0. 0.

Maximum number of entries: 5

BUNDLE_TEXT_COLOR

Description: Define the color indices for the TEXT bundle tables.

Type: Decimal

Default: 1 2 3 4 5

Maximum number of entries: 5

Polygon control

These keywords control the generation of polygons, closed objects which may be filled or hollow. The coordinates are encoded and formatted using the DEVICE_COORD definitions. If there is no polygon instruction on the device, then the polygon color instructions should be the same as the line color instructions.

POLYGON_INSTRUCTION_START

Description: If the device has a polygon instruction, then this field is defined. If the field is not defined, then line instructions are used to generate the outline of the polygon and fill style is hollow.

Type: String

Default: Null

Maximum number of entries: 40

POLYGON_INSTRUCTION_TERMINATOR

Description: Terminator for the hardware polygon instruction.

Type: ASCII characters

Default: Null

Maximum number of entries: 20

POLYGON_POINT_START

Description: Specifies the instruction to be issued prior to the encoding of each coordinate pair.

Type: String

Default: Null

Maximum number of entries: 20

POLYGON_POINT_TERMINATOR

Description: Specifies the instruction to be issued subsequent to the encoding of each coordinate pair.

Type: String

Default: Null

Maximum number of entries: 20

POLYGON_COLOR_INSTRUCTION_START

Description: The instruction to change polygon color.

Type: String

Default: Null

Maximum number of entries: 20

POLYGON_COLOR_INSTRUCTION_TERMINATOR

Description: Terminate the polygon color instruction.

Type: String

Default: Null

Maximum number of entries: 15

POLYGON_BACKGROUND_COLOR_INSTRUCTION_START

Description: The instruction to change the polygon color when using the background color. This is provided for those few devices that require special instructions for filling when using the background color.

Type: String

Default: Null

Maximum number of entries: 30

POLYGON_BACKGROUND_COLOR_INSTRUCTION_TERMINATOR

Description: Terminate the polygon color instruction when using the background color.

Type: String

Default: Null

Maximum number of entries: 15

POLYGON_MAXIMUM_POINTS

Description: Specifies the maximum number of points allowed in a hardware polygon instruction.

Type: Decimal

Default: 32767

Maximum number of entries: 1

POLYGON_SIMULATE

Description: Specifies whether a software polygon fill algorithm should be invoked by the translator. If this keyword is set to TRUE, then the fill algorithm will be invoked whenever the device has no hardware fill instruction, or when the number of vertices in the polygon exceeds POLYGON_MAXIMUM_POINTS.

Type: Logical

Default: TRUE

Maximum number of entries: 1

POLYGON_SIMULATION_SPACING

Description: This keyword specifies the interline spacing between fill lines for the software fill algorithm used in the translator. The number is in the range 0 to 32767. The software fill algorithm is invoked whenever POLYGON_SIMULATION_FLAG is TRUE and either there is no hardware fill instruction, or there is a hardware fill instruction but the number of vertices in the polygon exceeds POLYGON_MAXIMUM_POINTS.

Type: Decimal

Default: 0.9 * (32767. / ABS(DEVICE_COORD_UPPER_RIGHT_Y - DEVICE_COORD_LOWER_LEFT_Y)) [Truncated to decimal type.]

Maximum number of entries: 1

POLYGON_SIMULATION_TRUNCATION

Description: Specifies the length to be clipped off of fill lines in the software fill simulation. The number is in the range 0 to 32767. This keyword is potentially useful for film devices which show bright spots at line overlaps.

Type: Decimal

Default: 0

Maximum number of entries: 1

POLYGON_HATCH_SPACING

Description: This keyword specifies the interline spacing between fill lines for the hatch line patterns in polygon instructions when the interior style is hatch. The number is in the range 0 to 32767.

Type: Decimal

Default: 300

Maximum number of entries: 1

Raster control

These keywords control the translation of cell arrays. These keywords are utilized to interpret the CGM CELL ARRAY element (generated from the GKS CELL ARRAY function). For a detailed description of cell arrays, and how they are used, consult the GKS standard. Cell arrays are produced on devices having a raster display capability; not all output devices have such a capability. If an output device does not have a raster display capability, the keyword RASTER_SIMULATE should be set to FALSE; in this case a rectangle will be drawn around the cell array. If the output device has polygon fill, then cell arrays can be simulated if the RASTER_SIMULATE keyword is set to TRUE; see below for details.

The cell array can be drawn in any orientation or transposition by properly changing the lower left and upper right coordinates and perhaps interchanging the order of coordinate processing as defined in the RASTER_DATA_FORMAT. For example, if your device has lower left at (0, 0), and upper right at (1000, 1000), then you could draw the cell array upside down by setting the lower left coordinates at (0, 1000) and the upper right coordinates at (1000, 0).

Since cell arrays are subject to all of the coordinate transformations of GKS, it is possible that the cell array will not align with the coordinate axes. In this case, the translator will behave as if RASTER_SIMULATE were set to TRUE (that is, polygon fill will be used to create the cell array elements). A full implementation using raster scan lines for transformed cell arrays will appear in a later version of the translator. Also, the current translator is limited to accepting only horizontal scan line instructions. If an output device scans in the vertical direction, it will be necessary to set RASTER_SIMULATE to TRUE in the current version of the translator.

In this section, all coordinate addresses are assumed to be in the device's address space that allows for addressing individual pixels.

RASTER_SIMULATE

Description: If RASTER_SIMULATE is set to FALSE, then the translator will draw a rectangle around the cell array and nothing more. If an output device does not support scan line instructions, it can simulate cell arrays using polygon fill. If RASTER_SIMULATE is set to TRUE, polygon fill is used to draw each element of the cell array; if the device does not support polygon fill, then a rectangle will be drawn around each element of the cell array.

Note in particular that if RASTER_SIMULATE is set to TRUE, then any other RASTER keywords will be ignored, and polygon fill will be used.

Type: Logical

Default: FALSE (do not simulate scan line instructions)

Maximum number of entries: 1

RASTER_COORD_LOWER_LEFT_X

Description: The device X coordinate of the lower left corner of the drawing surface.

Type: Decimal

Default: Null

Maximum number of entries: 1

RASTER_COORD_LOWER_LEFT_Y

Description: The device Y coordinate of the lower left corner of the drawing surface.

Type: Decimal

Default: Null

Maximum number of entries: 1

RASTER_COORD_UPPER_RIGHT_X

Description: The device X coordinate of the upper right corner of the drawing surface.

Type: Decimal

Default: Null

Maximum number of entries: 1

RASTER_COORD_UPPER_RIGHT_Y

Description: The device Y coordinate of the upper right corner of the drawing surface.

Type: Decimal

Default: Null

Maximum number of entries: 1

RASTER_COORD_XOFFSET

Description: A device coordinate value added to X addresses before they are sent to the device. The translator attempts to center the plot based on the lower left and upper right values. When the coordinate space is asymmetric (X coordinate interval is not the same as the Y coordinate interval), offset values will help to place the frame properly. Another use of offset values is to move the display to a particular part of the display surface.

Type: Decimal

Default: 0 (zero)

Maximum number of entries: 1

RASTER_COORD_YOFFSET

Description: A device coordinate value added to Y addresses before they are sent to the device. See the description of RASTER_COORD_XOFFSET above.

Type: Decimal

Default: 0 (zero)

Maximum number of entries: 1

RASTER_COORD_XSCALE

Description: In some cases an X coordinate unit does not equal a Y coordinate unit. This keyword allows a scale factor to be introduced into the device coordinate computation stream. It will scale the X coordinates by the specified factor.

Type: Floating point

Default: 1.0

Maximum number of entries: 1

RASTER_COORD_YSCALE

Description: In some cases a Y coordinate unit does not equal an X coordinate unit. This keyword allows a scale factor to be introduced into the device coordinate computation stream. It will scale the Y coordinates by the specified factor.

Type: Floating point

Default: 1.0

Maximum number of entries: 1

RASTER_DATA_FORMAT

Description: This specifies the format used to convert the pixel color index values into device-specific pixel instructions. The input array to this format is a string of pixel color index values. Depending on the RASTER_HORIZONTAL_INSTRUCTION_START definition, the input array may contain a color index for every pixel, or it may be run-length encoded (an array containing pairs of values, the first element of the pair specifying a run length that is a count specifying a number of pixels, and the second element specifying a color index). Consult the Formatting and encoding coordinates section of this document.

In the example that follows, we describe the formatting required for the Tektronix 4107 for producing a RUNLENGTH WRITE instruction. The formatter receives an array of run-length encoded pixel values. The first value is the pixel count, and the second is the color index. Each set of run-length values must be formatted such that:

```
RUNCODE = number_of_pixels * (2 exp(n)) + color_index
```

where n is the number of bits per pixel used to specify the color index. In the following example, we use n=4, which allows the full 16 colors to be specified. The desired format table is:

```
--------------------------------------------------------------------
RASTER_DATA_FORMAT
/*  bit_start  bit_count  data_type  data_value
       1          11          1          10
      12           4          2           3
--------------------------------------------------------------------
```

Type: Decimal rows with four elements per row

Default: Null

Maximum number of entries: 40 (10 rows * 4 columns)

RASTER_DATA_ENCODING

Description: The encoding scheme used for the device raster data. See the section Formatting and encoding coordinates.

Type: Decimal in the range 0 to 5

Default: 0 (zero)

Maximum number of entries: 1

RASTER_DATA_FLOATING_INFO

Description: The mapping from fixed point (integer) to output floating point. This keyword is utilized when the RASTER_DATA_ENCODING is set to ASCII real (5). The keyword requires four floating point numbers. The first two describe the minimum and maximum data values received by the encoder. The final two describe the minimum and maximum floating point values sent to the device.

Type: Floating point

Default: Null

Maximum number of entries: 4

RASTER_VECTOR_COUNT_FORMAT

Description: The format used to create the device-specific raster vector counts. The formatter will receive only one input data word. The following example encodes a 16-bit data count:

```
----------------------------------------------------------------------------
RASTER_VECTOR_COUNT_FORMAT
/*  bit_start  bit_count  data_type  data_value
       0          16          1          15
----------------------------------------------------------------------------
```

Type: Decimal rows with four elements per row

Default: Null

Maximum number of entries: 40 (ten rows with four elements per row)

RASTER_VECTOR_COUNT_ENCODING

Description: The encoding scheme used for the device raster data. See the section Formatting and encoding coordinates.

Type: Decimal in the range 0 to 5

Default: 0 (zero)

Maximum number of entries: 1

RASTER_VECTOR_COUNT_FLOATING_INFO

Description: The mapping from fixed point (integer) to output floating point. This keyword is utilized when the RASTER_VECTOR_COUNT_ENCODING is set to ascii real (5). The keyword requires four floating point numbers. The first two describe the minimum and maximum data values received by the encoder. The final two describe the minimum and maximum floating point values sent to the device.

Type: Floating point

Default: Null

Maximum number of entries: 4

RASTER_HORIZONTAL_INSTRUCTION_START

Description: Defined if the output device has scan line commands; RASTER_SIMULATE must be FALSE to invoke the device raster scan commands.

Type: String

Default: Null

Maximum number of entries: 50 characters

RASTER_HORIZONTAL_INSTRUCTION_TERMINATOR

Description: Defined if the output device has scan line commands; RASTER_SIMULATE must be FALSE to invoke the device raster scan commands.

Type: String

Default: Null

Maximum number of entries: 50 characters

Fontcap files

Within the context of NCAR Graphics, a fontcap is a table describing a character font. This section describes how to generate fontcap files. Fontcap files have two formats: ASCII and binary---the ASCII file is created using a text editor, and the binary version is used for input to the CGM translators. The binary versions are generated automatically from the ASCII versions by using the fontcap preprocessor fontc. For details on how fontcaps fit into the overall picture of metafile translation, see the introductory section of this document, NCAR Computer Graphics Metafile translator.

In its ASCII form, the fontcap consists of a set of keywords followed by definitions for those keywords. A keyword must be contained on one line; typically a keyword will appear on one line, and the definitions for that keyword will appear on subsequent lines. Complete descriptions of all fontcap keywords appear later in this section.

To create or modify a fontcap, use any text editor that does not insert any of its own control characters into the file.

Definitions of fontcap keyword values are decimal (integer) values. All value definitions may range from null to the maximum count specified by the associated keyword. Keyword definitions are allowed a maximum of 80 columns per line, but they may cross line boundaries. A new line is equivalent to a space for separating definition entries.

Character digitizations are effected on a digitized grid. The size of this grid will depend on what the implementor of the font wants to use. The FONT CLASS keywords (described below) establish the digitization grid. Units in this grid are referred to as font units.

Important notes: Keywords in the fontcap must appear in the order they are described in this section. There are no default definitions. All keywords must be defined unless explicitly stated otherwise.

Character class

The following keywords define the collating sequence and the size of the characters in the font. There may not be any gaps in the collating sequence between the first character specified by CHARACTER_START and the last character specified by CHARACTER_END.

CHARACTER_START

Description: The decimal equivalent of the first character in the collating sequence.

Type: Decimal

Maximum number of entries: 1

CHARACTER_END

Description: The decimal equivalent of the last character in the collating sequence.

Type: Decimal

Maximum number of entries: 1

CHARACTER_WIDTH

Description: For mono-spaced fonts (see the definition of the FONT_TYPE keyword below), this is the width of each character in the font, including white space. For proportionally spaced fonts, this is the width of the widest character, including white space. All widths are measure in font units.

Type: Decimal

Maximum number of entries: 1

Font class

This class of keywords defines various character positions in the font. All fonts are assumed to be digitized on a fixed grid with the lower left corner equal to (0,0). Since this assumption requires that the left side of the font coordinate system is always 0, the obvious keyword FONT_LEFT is not required below. The keywords in this class closely follow the text model delineated in the GKS standard and the user is encouraged to consult that document for further explanation.

FONT_RIGHT

Description: The right side of the font coordinate system.

Type: Decimal

Maximum number of entries: 1

FONT_TOP

Description: The vertical top of the font coordinate system. In general, this value should be chosen so that abutting two characters vertically will allow appropriate white space between them.

Type: Decimal

Maximum number of entries: 1

FONT_CAP

Description: The vertical top of a capital letter given in the font coordinate system.

Type: Decimal

Maximum number of entries: 1

FONT_HALF

Description: The center, in the vertical direction, of a capital letter in the font coordinate system.

Type: Decimal

Maximum number of entries: 1

FONT_BASE

Description: The vertical base of a capital letter in the font coordinate system.

Type: Decimal

Maximum number of entries: 1

FONT_BOTTOM

Description: The vertical bottom of the font coordinate system. Descenders drop down to this level.

Type: Decimal

Maximum number of entries: 1

FONT_TYPE

Description: Flags the font type. Currently there are four legal types:

> **0**---mono-spaced stroked font.
> **1**---proportionally spaced stroked font.
> **2**---mono-spaced filled font.
> **3**---proportionally spaced filled font

The stroked fonts are graphically represented by sequences of line draws; the filled fonts are graphically represented by filled areas. If large stroked characters are drawn, the gaps between line draws will become obvious whereas filled characters do not have this drawback. However, in general it is much more expensive to draw filled characters.

Type: Decimal

Maximum number of entries: 1

Coordinate class

The coordinate class defines where each part of a character stroke is stored in an integer word (the packing instructions). The total of the coordinate definitions must fit in an integer word on the target machine. The paint class is not used except by the NCAR Dicomed camera translator and need not be defined unless the font is to be used by the Dicomed. Paint class defines a condition of where to start the actual beam exposure for each stroke. The start positions refer to the location in an integer word to start storing a bit string. The string is stored left to right. The start position aligns with the leftmost bit of the string to be stored. The indices are valued at 0 (zero) at the right of a word and increase to the left. For example, if it is desired to store the Y coordinates in the lower 9 bits of a word, then set COORD_Y_START to 8 and COORD_Y_LEN to 9.

COORD_X_START

Description: The starting point for storage of the X coordinate of a stroke.

Type: Decimal

Maximum number of entries: 1

COORD_Y_START

Description: The starting point for storage of the Y coordinate of a stroke.

Type: Decimal

Maximum number of entries: 1

COORD_PEN_START

Description: The starting point for storage of the pen up/down indicator.

Type: Decimal

Maximum number of entries: 1

COORD_X_LEN

Description: The number of bits in the X coordinate.

Type: Decimal

Maximum number of entries: 1

COORD_Y_LEN

Description: The number of bits in the Y coordinate.

Type: Decimal

Maximum number of entries: 1

COORD_PEN_LEN

Description: The number of bits in the pen indicator.

Type: Decimal

Maximum number of entries: 1

PAINT_BEGIN_START

Description: The starting point for storage of the paint begin indicator.

Type: Decimal

Maximum number of entries: 1

PAINT_END_START

Description: The starting point for storage of the paint end indicator.

Type: Decimal

Maximum number of entries: 1

PAINT_BEGIN_LEN

Description: The number of bits in the paint begin indicator.

Type: Decimal

Maximum number of entries: 1

PAINT_END_LEN

Description: The number of bits in the paint end indicator.

Type: Decimal

Maximum number of entries: 1

Character stroke class

This class defines the characters in their proper collating sequence. It has only one keyword, which delimits each character.

CHAR

Description: This keyword defines the strokes (for FONT_TYPEs equal to 0 or 1), or polygons (for FONT_TYPEs equal to 2 or 3) for each character in the font. Each character must be in its proper position in the collating sequence and must be preceded by a CHAR keyword. The definition of a character will differ depending on the FONT_TYPE. If the font is proportionally spaced, then the first two entries in the CHAR value define the character left and character right values which define the horizontal limits of the character body. For proportionally spaced fonts, these character left and character right values should be defined so that two horizontally contiguous characters will have the appropriate white space between them. For filled fonts, the "paint start" and "paint end" values have no meaning and consequently should not be specified. Examining some example fonts in the package should help clarify the definition of the CHAR values. The PEN value flags a move or draw for stroked fonts, and flags begin polygon or continue polygon for filled fonts.

A single data item in a CHAR value will have five components (only the first three for filled fonts).

X-coordinate
: The X coordinate given in font coordinate space.

Y-coordinate
: The Y coordinate given in font coordinate space.

PEN
: The pen. For stroked fonts, 0 indicates pen up (a move); 1 indicates pen down (draw). For filled fonts, 0 indicates the start of a new polygon, and 1 indicates to add the current point to the current polygon.

Paint Start bit
: The draw at start of vector bit. Interpreted same as PEN, must be either 0 or 1. Only effective for the NCAR Dicomed translator, and must not be specified for filled fonts.

Paint End bit
: The draw at end of vector bit. Interpreted same as PEN, must be either 0 or 1. Only effective for the NCAR Dicomed translator, and must not be specified for filled fonts.

All five components must be included for each stroke in a stroked font, but only the first three items must appear for each data item in a filled font. The paint bits may be zero or one at any time unless you intend to use the font for the Dicomed.

Presented below are examples of two characters as defined in two different fontcap files. The first character "#" is defined as a member of a mono-spaced font, and the second character "A" is defined as a member of a proportionally spaced font. Note that for the proportionally spaced font, the first two parameters in the value specification specify the character left and character right values. In the first example, the X and Y lengths are six bits. In the second example, the X and Y lengths are eight bits. The PEN and PAINT are one bit.

```
----------------------------------------------------
CHAR
/* #
/*  x_coord  y_coord  pen  paint_st  paint_ed
       7        33      0      0         0
       5        18      1      1         1
      12        18      0      0         0
      14        33      1      1         1
```

```
      18       29      0      0         0
       3       29      1      1         1
       1       22      0      0         0
      16       22      1      1         1
-----------------------------------------------------
```

```
--------------------------------------------------------------------
CHAR
/* A
/*     character left  character right
            122              132
/*        x_coord          y_coord      pen  paint_st  paint_ed
            127              132         0      0         0
            123              123         1      1         1
            127              132         0      0         0
            131              123         1      1         1
            125              126         0      0         0
            129              126         1      1         1
--------------------------------------------------------------------
```

Type: Decimal

Maximum number of entries:

- For X coordinates, Y coordinates, PEN, and PAINT, the user defines the range of values allowed with the COORDINATE CLASS keywords.
- There are a maximum of 300 strokes allowed per character.
- A maximum of 100 characters may be defined.
- The total stroke count cannot exceed 5121.

Font tables

There are currently 20 supported fontcaps. The tables on the following pages list all the supported fonts. The left column in the tables gives the ASCII character used in a TEXT element, and the other columns present the character as it will be stroked. The association of the Hershey fonts with the font numbers is given by:

```
-----------------------------------------------------------
GKS font      Font name
number
  1           DEFAULT
 -2           HERSHEY:CARTOGRAPHIC_ROMAN
 -3           HERSHEY:CARTOGRAPHIC_GREEK
 -4           HERSHEY:SIMPLEX_ROMAN
 -5           HERSHEY:SIMPLEX_GREEK
 -6           HERSHEY:SIMPLEX_SCRIPT
 -7           HERSHEY:COMPLEX_ROMAN
 -8           HERSHEY:COMPLEX_GREEK
 -9           HERSHEY:COMPLEX_SCRIPT
-10           HERSHEY:COMPLEX_ITALIC
-11           HERSHEY:COMPLEX_CYRILLIC
-12           HERSHEY:DUPLEX_ROMAN
-13           HERSHEY:TRIPLEX_ROMAN
-14           HERSHEY:TRIPLEX_ITALIC
-15           HERSHEY:GOTHIC_GERMAN
```

```
-16         HERSHEY:GOTHIC_ENGLISH
-17         HERSHEY:GOTHIC_ITALIAN
-18         HERSHEY:MATH_SYMBOLS
-19         HERSHEY:SYMBOL_SET1
-20         HERSHEY:SYMBOL_SET2
-21         NCAR:HELVETICA (filled font for Plotchar)
-22         NCAR:HELVETICA-BOLD (filled font for Plotchar)
-25         NCAR:TIMES-ROMAN (filled font for Plotchar)
-26         NCAR:TIMES-BOLD (filled font for Plotchar)
-29         NCAR:COURIER (filled font for Plotchar)
-30         NCAR:COURIER-BOLD (filled font for Plotchar)
-33         NCAR:GREEK (filled font for Plotchar)
-34         NCAR:MATH-SYMBOLS (filled font for Plotchar)
-35         NCAR:TEXT-SYMBOLS (filled font for Plotchar)
-36         NCAR:WEATHER1 (filled font for Plotchar)
-37         NCAR:WEATHER2 (filled font for Plotchar)
----------------------------------------------------------
```

NCAR Computer Graphics Metafile

This section describes the NCAR implementation of the Computer Graphics Metafile (CGM) standard. For a general description of metafiles and of the CGM, read the introductory section of this document, "NCAR Computer Graphics Metafile translator."

The elements of the NCAR metafile comprise a proper subset of the elements of the American National Standards Institute (ANSI) CGM standard, as described in the document:

> *ANSI X3.122-1986*
> *Information Processing Systems*
> *Computer Graphics*
> *Metafile for the Storage and Transfer of Picture Description Information*

This document is available from:

> ANSI
> 1430 Broadway
> New York, NY 10018
> Phone: (212) 354-3300

The CGM is also an International Standards Organization (ISO) standard.

Record formatting and NCAR datatypes

The NCAR CGM uses the binary encoding described in the CGM standard. The CGM standard only describes the bit stream that comprises a metafile; it does not describe how the bit stream is divided into physical records.

Experimentation at NCAR has indicated that fixed-length records have great portability advantages when the files will potentially be used on diverse computer systems. It is advantageous to specify the record length as an integral number that is divisible by the word lengths of as many computers as possible. A value of 11520 bits, or 1440 8-bit bytes, serves as a reasonable compromise between the requirements and capabilities of a range of systems from microcomputers to supercomputers. This value was previously used in the record structure of the old (pre-CGM) NCAR metacode.

The NCAR implementation of the CGM allows non-CGM records to be mixed with CGM records on certain occasions. To flag CGM or non-CGM records, a 4-byte control field is reserved at the beginning of each record. The first 16 bits of this control field is a data count that indicates the number of bytes in the remainder of the record that actually contain useful data (this count does not include the first four bytes, so the largest legal value it can have is 1436 decimal). The next four bits in the control field comprise the datatype identifier field. Currently defined datatypes include header (0100 binary), NCAR-formatted printer (1000 binary), pre-CGM NCAR metacode (0010 binary), and NCAR CGM (0011 binary).

The single bit following the 4-bit datatype identifier is used as a "new-frame bit." When the new-frame bit is set, the data record in which it is contained is the first record of a new picture. This formatting feature does not prompt any graphical action; it is the BEGIN PICTURE and END PICTURE instructions of the CGM standard that identify the boundaries of the graphical contents of a picture. You will find the new-frame bit very useful for non-interpretive processing software such as metafile editors that extract whole frames from metafiles, split metafiles, and concatenate metafiles.

The new-frame bit as defined above was also defined for the pre-CGM NCAR metacode. The two bits following the new-frame bit are now defined for the new NCAR CGM only. The first of the newly defined bits is the "begin metafile" bit, which marks the first record of a metafile and hence the record containing the metafile descriptor. The bit that follows the "begin metafile" bit is the "end metafile" bit, which declares that the record is the last record of a metafile and that it contains an END METAFILE instruction. These two bits allow multiple metafiles to reside on a single physical medium, and to be easily recognized by non-interpretative processors such as editors.

The control bits are summarized in their order as follows:

1-16: the byte count for the record
17-20: the datatype flag
21: the new-frame bit
22: the begin metafile bit
23: the end metafile bit
24-32: currently undefined

The zero-byte terminating record of the pre-CGM metafile is of little use, and, in fact, has caused problems in the design logic of metafile editors. It is no longer necessary to terminate an NCAR metafile with a zero-byte count record. The metafile is a single binary file terminated with an EOF (End Of File) mark.

Although instructions may cross record boundaries, an individual operand component such as a coordinate or a color index may not be split across record boundaries. The 16-bit alignment requirement in the CGM standard makes it impossible to split operand components across record boundaries, since the default precision value of the metafile is 16 bits. This default value applies to coordinate precision, color index precision, and so on. Splitting operand components across record boundaries would be possible if precision were increased beyond 16 bits. All instructions start on 16-bit boundaries.

The CGM defines a metafile descriptor that contains no pictorial information, but includes descriptive information that aids in interpreting the metafile. The NCAR metafile descriptor consists of one or more fixed-length data records that look like any other metafile data records. There is only one metafile descriptor, and it occurs at the beginning of the metafile. The new-frame bit of the 32-bit control field is set to zero in metafile descriptor records. The metafile descriptor in the NCAR CGM is that sequence of contiguous records at the beginning of the metafile up to (but not including) the first record with the new-frame bit set.

Supported and unsupported elements

As previously mentioned, the NCAR CGM is a proper subset of those elements mentioned in the CGM standard. The following tables detail which elements are, and which elements are not, supported in the NCAR CGM as of August 1987. "Supported" means that the elements can be generated by the NCAR GKS package, and they are interpreted by the NCAR CGM translator.

Class	Supported Elements	Unsupported Elements
0	no-op	
	BEGIN METAFILE (1)	
	END METAFILE	
	BEGIN PICTURE	
	BEGIN PICTURE BODY	
	END PICTURE	
1	METAFILE VERSION	COLOR VALUE EXTENT
	METAFILE DESCRIPTION	CHARACTER SET LIST
	VDC TYPE (2)	CHARACTER CODING
	INTEGER PRECISION	ANNOUNCER
	REAL PRECISION	
	INDEX PRECISION	
	COLOR PRECISION (3)	
	COLOR INDEX PRECISION (3)	
	MAXIMUM COLOR INDEX	
	METAFILE ELEMENT LIST	
	METAFILE DEFAULTS REPLACEMENT	
	FONT LIST	
2	COLOR SELECTION MODE (4)	SCALING MODE
	LINE WIDTH SPECIFICATION MODE (5)	EDGE WIDTH SPECIFICATION MODE
	MARKER SIZE SPECIFICATION MODE (5)	
	VDC EXTENT	
	BACKGROUND COLOR	
3	VDC INTEGER PRECISION	AUXILIARY COLOR
	VDC REAL PRECISION	TRANSPARENCY
	CLIP RECTANGLE	
	CLIP INDICATOR	
	(1) Only a single metafile may be contained in a single output medium.	
	(2) Only the integer type is supported.	
	(3) Only 8-bit precision is supported	
	(4) Only indexed mode is supported.	
	(5) Only scaled mode is supported.	
4	POLYLINE	RESTRICTED TEXT
	DISJOINT POLYLINE	APPEND TEXT
	POLYMARKER	POLYGON SET
	POLYGON	RECTANGLE

	TEXT	CIRCLE
	CELL ARRAY	CIRCULAR ARC 3 POINT
		CIRCULAR ARC 3 POINT CLOSE
		CIRCULAR ARC CENTER
		CIRCULAR ARC CENTER CLOSE
		ELLIPSE
		ELLIPTICAL ARC
		ELLIPTICAL ARC CLOSE
5	LINE TYPE	LINE BUNDLE INDEX
	LINE WIDTH	MARKER BUNDLE INDEX
	LINE COLOR	TEXT BUNDLE INDEX
	MARKER TYPE	CHARACTER SET INDEX
	MARKER SIZE	ALTERNATE CHARACTER SET
	MARKER COLOR	INDEX
	TEXT FONT INDEX	FILL BUNDLE INDEX
	TEXT PRECISION	PATTERN INDEX
	CHARACTER EXPANSION	EDGE BUNDLE INDEX
	FACTOR	EDGE TYPE
	CHARACTER SPACING	EDGE WIDTH
	TEXT COLOR	EDGE COLOR
	CHARACTER HEIGHT	EDGE VISIBILITY
	CHARACTER ORIENTATION	FILL REFERENCE POINT
	TEXT PATH	PATTERN TABLE
	TEXT ALIGNMENT	PATTERN SIZE
	INTERIOR STYLE	ASPECT SOURCE FLAGS
	FILL COLOR	
	HATCH INDEX	
	COLOR TABLE	
6	ESCAPE	
7		MESSAGE
		APPLICATION DATA

Appendix A: Supported graphcaps in Version 4.*x*

This table lists all the graphcaps supported in the Version 4.*x* release of NCAR Graphics.

CTXT	Produces a human-readable ASCII dump of the input.
a60	Abekas a60 raster image format.
adm5	ADM5 with the DEC RG1500 graphics board.
aed.a	AED512 in ASCII mode.
aed.b	AED512 in binary mode.
avs	Application Visualization System raster image format.
balsml	HI DMP-29 in small chart mode.
ditroff	Device-independent troff.
form	A blank form for user-defined graphcap definitions.
hdf	NCSA's HDF (Hierarchical Data Format) formatted raster file.
hp2648a	HP2648A graphics terminal.
hp150	HP150 personal computer.
hp7475a	HP7475A six-pen plotter.
hp7510a	HP7510A color film recorder.
hpgl	Basic support for the HP-GL language in portrait mode.
hpgl.land	Basic support for the HP-GL language in landscape mode.
hppcl	Produces files for the HP LaserJet family of printers (LaserJet, LaserJet Plus, LaserJet Series II, LaserJet 500 Plus, etc.); the **ctrans** options **-dpi** and **-landscape** can be

```
                used for various resolutions and picture positioning.
imagen          Graphcap for the IMAGEN 8/300 laser printer in
                graphics landscape mode.
imagen.port     Graphcap for the IMAGEN 8/300 laser printer in
                graphics portrait mode.
nrif            NCAR Raster Interchange Format formatted raster file.
pc.mono         Drives PCPLOT on IBM and compatible PCs.
ps.color        Color PostScript graphcap (portrait mode).
ps.land.color   Color PostScript graphcap (landscape mode).
ps.land.mono    Black and white PostScript graphcap (landscape mode).
ps.mono         Black and white PostScript graphcap (portrait mode).
qms800          QMS800 laser printer.
r6211           Ramtek 6211 in Tektronix compatible mode.
s100            Selinar HiREZ100 graphics terminal.
sgi             Silicon Graphics raster image format.
sun             Sun formatted raster file.
t4010           Tektronix 4010 and 4012.
t4025           Tektronix 4025 graphics terminal.
t4105           Tektronix 4105 color graphics terminal.
t4107           Tektronix 4107 color graphics terminal.
t4107.seg       Tektronix 4107 color graphics terminal. All graphics
                instructions are stored in local memory (a segment) for
                manipulation with functions defined in the hardware.
t4115           Tektronix 4115 color graphics terminal.
t4115.seg       Tektronix 4115 color graphics terminal. All graphics
                instructions are stored in local memory (a segment) for
                manipulation with functions defined in the hardware.
tal1590         Talaris 1590 plotter.
tekalike        Tekalike program on the Apple Macintosh.
versaterm       Versaterm program on the Apple Macintosh.
vt100           VT100 with Digital Engineering VT640 retrofit.
vt125           VT125 graphics terminal.
vt220           VT220 with Selinar SG220 graphics retrofit board.
vt330           VT330 graphics terminal.
vt340           VT340 graphics terminal (returns with white foreground
                and black background).
vt340w          VT340 graphics terminal (returns with black background
                and white foreground).
X11             X Window System interface.
xwd             X11 xwd formatted raster file.
--------------------------------------------------------------------------
```