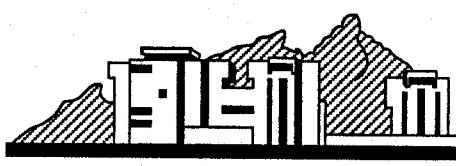
NCAR/TN-342+STR

NCAR GRAPHICS GENERIC PACKAGE INSTALLER'S GUIDE

VERSION 3.00a

FRED CLARE



SCIENTIFIC COMPUTING DIVISION
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH

Acknowledgments

The author would like to acknowledge John Humbrecht as the author of the original draft of the "NCAR Computer Graphics Metafile Translator." I also want to thank Dr. Hiroshi Akima for allowing us to include the BIVAR software in Version 3.00 of NCAR Graphics. BIVAR was written as part of Dr. Akima's work for the National Telecommunications and Information Administration, Institute for Telecommunication Sciences, within the U.S. Department of Commerce. Thanks to Dr. Alan Cline of the University of Texas at Austin for allowing us to use selected routines from his package called FITPACK. And thanks to the many users of the previous documentation who have made valuable criticisms and comments.

Thanks to Nancy Dawson for her editorial and formatting suggestions, to Christine Guzy for production assistance, and to Jacque Marshall for word processing.

November 9, 1990

Addendum to NCAR Graphics Generic Package Installer's Guide

1. On page 19, description for "FUNCTION ISHIFT(IWORD,N)": After "...then the vacated positions are undefined." please add the sentence "NCAR Graphics does not care what the vacated positions in a right shift are."

2. On pages 68 and 69, replace the "Device Window" section with:

In the default environment CGM interpreters map the entirety of the region specified in the CGM VDC EXTENT element onto the largest rectangle on the viewing surface such that aspect ratio is preserved. This section describes keywords for specifying a subregion of the VDC space to be mapped onto the largest rectangle on the viewing surface such that aspect ratio is preserved. To define the keywords, imagine a virtual rectangular space having coordinates from 0 to 32767 in both the X and Y directions being mapped *onto* the VDC space. If the VDC space is not square, a unit in the X direction of the virtual space will not equal a unit in the Y direction. The following keywords are used to specify a subregion of the virtual space so that the associated subregion of the VDC space is used for mapping onto the display surface. For example, if the VDC space is given by (0,0) to (100,200) and the subregion of the virtual space is defined by (0,0) to (16384,16384) then the VDC region (0,0) to (50,100) would be mapped onto the display surface in a manner such that aspect ratio is preserved.

DEVICE_WINDOW_LOWER_LEFT_X

Description: The X coordinate of the lower left corner of the virtual space 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 virtual space 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 virtual space 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 virtual space in the rnage 0 to 32767.

Type: Decimal

Default: 32767

Maximum number of entries: 1

Contents

Section I: Important Information

Section I: Important Information

Important Information

The material in this manual is intended to be used by an experienced programmer during the initial installation of NCAR Graphics, Version 3.00. Users at your site will need to have both of the following user's manuals:

- The *NCAR Graphics Guide to New Utilities, Version 3.00*, October 1989. This manual documents the seven utilities that are new in Version 3.00.
- The *NCAR Graphics User's Guide, Version 2.00*, August 1987. This manual documents all the previous utilities that are part of the Version 3.00 release.

Both of these manuals were sent to you with the distribution tape. You will need to refer to both manuals while you are installing NCAR Graphics. There are several references to these manuals throughout the Installation Guide you are now reading. If the reference is to both manuals, they will be referred to collectively as the User's Guides. If the reference is to only one manual, the full title will be given.

Before You Begin the Installation

Before you actually begin the installation, do the following:

1. Read the Introduction chapter in *NCAR Graphics Guide to New Utilities, Version 3.00.* Two sections are of particular relevance, since they tell users what to expect to have available. These sections are "The Code for the NCAR Graphics Package" and "How to Begin Using NCAR Graphics."
2. Finish reading this section of the Installer's Guide.
3. If you are installing the UNIX version, follow the instructions in the "Release Notice: NCAR Graphics — UNIX Version" document that came with your tape.
4. If you are installing the VMS version, follow the instructions in the "NCAR Graphics VMS Version 3.00 Installation Guide," which came with your tape.
5. If you are installing the generic version, proceed to Section II for installation details.

How This Manual is Organized

The *NCAR Graphics Generic Package Installer's Guide, Version 3.00* has five sections:

1. Important Information. This is the section you are currently reading.
2. NCAR Graphics Installation Directions. This section is a roadmap for implementation of the entire package and it is intended to be used by the initial installer of the package.
3. Release Contents. This section gives brief descriptions of all files on the generic package distribution tape.
4. NCAR Computer Graphics Metafile Translator. This section gives background material on metafiles, and describes the NCAR Metafile Translator, complete with implementation instructions.

5. NCAR Computer Graphics Metafile (CGM). This section describes the record formatting of NCAR's implementation of the CGM as well as which elements of the CGM are supported and which elements are not supported.

Installation Consulting

If you have questions related to installing the package, call NCAR Graphics Installation at (303) 497-1309, or send e-mail to ncargfx@ncar.ucar.edu. Please note that this phone number and e-mail address are for installation questions only.

Usage Consulting through NCAR Graphics Site Representatives

Each site ordering the NCAR Graphics package must designate an official "NCAR Graphics site representative" when the package is ordered. The site representative serves as the first line of consulting aid for all users at the site. If the site representative cannot answer a user's question, the representative may call the SCD Consulting Office at (303) 497-1278 for help. Each site may designate two representatives.

Please publicize the name and phone number of your site representative so that users know how to get assistance. To update site representative information, contact

Graphics Information
NCAR Scientific Computing Division
P.O. Box 3000
Boulder, CO 80307-3000
(303) 497-1201
scdinfo@ncar.ucar.edu

Licensing and Distribution Information

NCAR Graphics is a copyrighted software package developed and distributed by the Scientific Computing Division (SCD) at the National Center for Atmospheric Research (NCAR). NCAR Graphics is licensed to nonprofit institutions by NCAR and to profit-making institutions by the University Corporation for Atmospheric Research (UCAR) Foundation. Through the UCAR Foundation, developers' licenses are available to software vendors and computer manufacturers who wish to make value-added enhancements. Versions of NCAR Graphics are available for UNIX, VMS, and generic operating systems. Additional copies of NCAR Graphics manuals are also available for purchase separately. For licensing, pricing, and distribution information, contact

Graphics Information
NCAR Scientific Computing Division
P.O. Box 3000
Boulder, CO 80307-3000
(303) 497-1201
scdinfo@ncar.ucar.edu

Section II: NCAR Graphics Installation Directions

Section II: NCAR Graphics Installation Directions

NCAR Graphics Installation Directions

This guide is designed to provide step-by-step guidance for the implementation of the generic NCAR Graphics package. It is anticipated that the user of this guide will be an experienced programmer who is familiar with the hardware and software of the target computer(s) and graphics output device(s). This guide does not address the details associated with implementation on particular computers, but it is intended to provide instruction in the major steps required in implementing the generic package on any system. If the target computers are running either the UNIX or VMS operating systems, then the installer should have the version of the package specifically tailored for those systems as well as supplemental documentation for those systems. Ordering information is in the "Important Information" section of this manual. The generic package contains only Fortran code (except for some examples of support routines written in C). The UNIX and VMS versions of the package have certain C language implementations additional to the code in the generic package.

This guide describes an implementation procedure that begins at the lowest levels of the system and proceeds to the highest levels. We assume that you have already read the Introduction section in the *NCAR Graphics Guide to New Utilities, Version 3.00*, before proceeding with the installation.

Package Contents

A detailed listing of the contents of the NCAR Graphics package is given in the Release Contents, which is Section III of this manual. A list of the filenames in the sequence as they appear on the distribution tape is contained in the first file on the tape, FNAMES, and is also attached to the cover letter you received with the distribution tape. These names are the same as those that appear throughout all sections of this manual, so it is important that the correct name be associated with the correct file during the installation.

Implementing the System

The major components involved in a full implementation of NCAR Graphics are listed below:

- Machine-specific support routines,
- a GKS 0A package,
- SPPS — The NCAR System Plot Package Simulator,
- a Computer Graphics Metafile (CGM) translator, and
- higher-level utilities and demonstration drivers.

Each of these components is fully explained in the following instructions.

Partial Implementations

The basic assumption in this manual is that all components of the package will be implemented, including NCAR's GKS level 0A package. NCAR's GKS package contains some of the capabilities of level 2A, which are needed if GFLASH or STITLE are going to be used at your site. It is possible to implement certain components of the package independently.

If you are using a commercial GKS package and simply want to run some, or all, of the higher-level utilities (AUTOGRPH, CONPACK, EZMAP, and so on) on top of the commercial GKS package, then it would be necessary to implement only SPPS, the desired utility (or utilities) and certain of the low-level support routines mentioned in the "Step 1" section below. To be certain that you have implemented the appropriate low-level support routines, you can, of course, implement them all. Another approach would be to load SPPS and the desired utility and see what unsatisfied externals are listed, and then implement just those functions that appear in this list.

The implementation of the NCAR Computer Graphics Metafile translator may be carried out independently from the rest of the package, excepting that certain of the low-level support routines mentioned in Step 1 below are required. When discussing the metafile translator in this document, we are referring specifically to the Fortran version of that translator. A C language implementation is available as part of the NCAR View package, which is included in the UNIX distribution. If you want to implement only the Metafile Translator, then consult the section of this manual titled "The NCAR Computer Graphics Metafile Translator"; that section has complete installation instructions.

The following discussion provides a more detailed description of the steps involved in implementing each of the major components of the package in a full implementation. Implementation should proceed in the order described, except for the implementation of the metafile translator (Step 4 below), which could be implemented immediately after Step 1, if desired. The suggested order for implementation insures that you will have a couple of metafiles available for testing the translator by the time it is implemented.

For each step in the implementation, a test is referenced for verifying correctness. If any test plots are referenced, examples of them may be found by using the table on page 13.

Implementation Steps

Step 1 — Implementation of Lower-level Support Routines

The package is designed to be portable, but there are certain machine-specific details that are required. All of these machine-dependent requirements have been isolated in several subroutines. It will be necessary to provide these subroutines before proceeding. The package requires the implementation of the following support routines:

GBYTES	ISHIFT
G01MIO	I1MACH
IAND	R1MACH
IOR	SBYTES

These routines are referred to as "low-level support routines" throughout this manual. Complete functional descriptions for these routines appear in Appendix A at the end of this section. Except for G01MIO, examples of implementations of these subroutines are given in file LOCAL on the distribution tape. A Fortran implementation of the support routine G01MIO

is contained in file BWI. G01MIO is the basic I/O routine for NCAR's GKS package, and it is not used anywhere else but in that package. The version of G01MIO supplied does a Fortran open to file GMETA which is where the metafile is written. Users of NCAR's GKS package may change the name of the output metafile by a call to the GKS entry GESC with a function ID of -1391 and the desired filename stored in the first array element of the input data record (the data record should be blank filled to the right of the file name). The subroutines in LOCAL and G01MIO **are examples only**. These examples may help you, and some may actually run on your machine, but care must be taken to insure that the implemented routines satisfy the functional descriptions as given in Appendix A at the back of this section. Some of the examples given are coded in C. If you do not have a C compiler available to you, then obviously these examples will be of little help to you, and you will have to provide your own codings in Fortran, or any other language you have access to.

There is a test file for implementations of the low-level routines in file IMPLTEST on the distribution tape. In that file, there is a PROGRAM TLOCAL, which can be used to test the implementations of IOR, IAND, ISHIFT, GBYTES, and SBYTES. Load and execute TLOCAL together with the implementations of the support routines. Note that TLOCAL has machine-dependent parameters that must be set by the user. Read the prologue documentation in the code for TLOCAL for implementation instructions. Success or failure messages will be issued to Fortran unit 6. There are no tests for I1MACH and R1MACH, but the success of the TLOCAL test depends on proper implementation of I1MACH and R1MACH. Constants for I1MACH and R1MACH for a large number of computers appear in the comment cards of I1MACH and R1MACH as they appear on the distribution tape in file LOCAL. If constants for your host computer appear there, simply uncomment the appropriate cards for your implementation of I1MACH and R1MACH. Otherwise be very careful to implement I1MACH and R1MACH correctly since there is no test for them. The support routine G01MIO is used only by NCAR's GKS package, and no test for it is provided in TLOCAL.

Since many of the low-level support routines are executed frequently throughout the package, efficient versions are desirable. There are portable Fortran versions of GBYTES and SBYTES in file LOCAL on the distribution tape; they are very slow. GBYTES and SBYTES are used primarily in the CGM translator; machine language versions of these routines could greatly speed up the translator. Machine language versions of IAND, IOR, and ISHIFT are also desirable.

Some entries from the file SUPPORT on the distribution tape are required by the test routines.

Step 2 — Implementation of NCAR's GKS 0A Package

NCAR's implementation of the ANSI (American National Standards Institute) GKS graphics standard is contained in files AWI and BWI on the distribution tape. (AWI stands for "Above Workstation Interface" and BWI stands for "Below Workstation Interface.")

GKS can be implemented at various levels of complexity. The NCAR implementation is a level 0A implementation, with enhancements from level 2A that are necessary to support the new utilities GFLASH and STITLE. The NCAR GKS package has approximately the same functionality as the NCAR System Plot Package in pre-GKS versions of the NCAR Graphics Package.

AWI intercepts GKS level 0A calls and codes the arguments into a common block; AWI invokes BWI when appropriate. BWI generates the output Computer Graphics Metafile (CGM) from the data stored in the common block in AWI. The CGM is also an ANSI standard; for details, see the "NCAR Computer Graphics Metafile Translator" and the "NCAR Computer Graphics Metafile" section of this manual.

If you want to drive an output device directly and bypass the generation of the CGM, it would be possible to replace BWI with such a direct device driver. This could involve a significant amount of work. The NCAR GKS package as supplied on the distribution tape can be used as the support GKS package for all the other package components. However, those sites having other GKS software already running on the target computer may elect to interface the utilities with their existing capabilities. The NCAR GKS 0A package allows for only one active workstation (we use the term "workstation" in this document in its sense as defined in the GKS standard), and that is an MO (Metafile Output) workstation. In the NCAR GKS package, the CGM workstation is defined as a workstation having workstation type 1. If you load and execute the subroutines contained in AWI, BWI, the low-level support routines, and a GKS application program, then what should be produced is a CGM Metafile in file GMETA.

A test file for the output primitives of GKS is contained in file IMPLTEST on the distribution tape. In file IMPLTEST is contained PROGRAM TGKS0A. If this program is extracted from file IMPLTEST and loaded and executed with AWI, BWI, and the support routines from Step 1, then a metafile will be produced in file GMETA. Since you have yet to implement the CGM translator, you cannot draw the plot contained in GMETA. However, octal and hex dumps of the metafile are provided in Appendix B at the end of this section. Compare a dump of your metafile against those in Appendix B for correctness. Save the metafile for plotting once the translator has been implemented, and compare the plot with the plot on page 623 of the *NCAR Graphics User's Guide, Version 2.00.* Unless your output device supports filled polygons and cell arrays, those objects may appear on the plot with only their perimeters drawn (the minimal support level required by GKS.) The Fortran translator does contain logic for simulation of filled areas in software. See the "NCAR Computer Graphics Metafile Translator" section for details.

On EBCDIC computers (most IBM equipment), the EBCDIC character set must be converted to ASCII prior to being written to the CGM because the CGM standard requires ASCII characters. This conversion is achieved using the routine GKASC in the file BWI. As distributed, this routine does nothing except make its output equal to its input. For EBCDIC computers, a conversion table should be put into this routine for providing the EBCIDC to ASCII conversion.

Step 3 — Implementation of SPPS

The implementation of SPPS (the System Plot Package Simulator) should be straightforward — simply compile it and load it with a GKS package. The higher-level utilities depend heavily upon SPPS, so it must be loaded whenever a higher-level utility is used.

There are two tests for SPPS contained in file IMPLTEST on the distribution tape. The simpler is PROGRAM TPLOT. Loading PROGRAM TPLOT from IMPLTEST together with the support routines from Step 1, and the NCAR GKS package will produce a metafile. One very simple plot is produced from this test. Since you have not yet implemented the translator, the only way to test the correctness of these plots is to compare your metafile with the octal and hex dumps which appear in the comment cards in PROGRAM TPLOT. There is a more complete test of SPPS contained in file IMPLTEST; this is PROGRAM

TSPPS. This tests every SPPS entry — even the non-plotting entries. You will want to postpone running this test until you have implemented the CGM translator so that you can actually examine the plots produced for correctness. The plots from all of the test packages appear in the User's Guide.

Step 4 — Implementation of the NCAR CGM Translator

The NCAR CGM translator is a program that translates a CGM into instructions specific to a given output device. It is table-driven in that there is one master translator, and for any given output device the translator configures itself appropriately after reading in a user-supplied description table for the given device. It should be emphasized that the device description tables apply only to devices that allow user access to the native command set — most entries in the table are commands specific to the output device. If the interface to a graphics device is a software interface, such as the subroutine-call interface to many workstations, then modifications to the translator code will be required. There is a file of device description tables on the distribution tape in GRAPHCAP. This file contains tables for many common output devices.

The translator is the most complex component of the package to implement. After the implementation of the support routines in Step 1, the translator could be implemented. The translator does not depend on any of the other components of the package for its implementation.

Complete details on the installation of the translator are contained in the section of this manual titled the "NCAR Computer Graphics Metafile Translator." That section contains its own installation guide.

After successful implementation of the translator, all previously mentioned test files can be plotted. This will serve as a further test for those files as well as a test for the translator itself.

Step 5 — Implementation of the Higher-level Utilities

The higher-level utilities can be implemented on an as-needed basis — by-and-large the implementation of one utility does not depend on that of another. The implementation of a given utility simply amounts to compiling it and loading it with the proper libraries. For example, to call AUTOGRPH from a user application, load the user application together with AUTOGRPH, SPPS, a GKS package, and the support routines from Step 1.

Version 3.00 contains upgrades and replacements for previous versions of certain utilities. CONPACK upgrades and replaces the old contouring family consisting of CONREC, CONRCQCK, and CONRCSPR. New implementations of the package should contain only CONPACK and not the previous utilities. The previous utilities are provided on the tape for those who may want to maintain compatibility with previous versions of the package. Also, BIVAR (in the EMISC file on the distribution tape) shows how to convert randomly spaced data to gridded data for use with CONPACK. We recommend that users use BIVAR in combination with CONPACK, rather than using CONRAN, CONRAQ, or CONRAS. PLOTCHAR is a replacement for PWRITX and PWRITY. It offers a greatly simplified user interface.

Several of the utilities depend on subroutines in the file SUPPORT on the distribution tape.

For each of the higher-level utilities there is a demonstration driver supplied on the distribution tape. These demonstration drivers invoke the given utility to produce a typical plot (or plots). Each demo driver is a subroutine with a single argument — an error indicator that is returned as zero for success and one for failure. In most cases, the drivers for the demo subroutines are as simple as

```
CALL OPNGKS
CALL T name
CALL CLSGKS
```

There are five exceptions:

EZMAP	requires the map database be opened on Fortran logical unit 1.
EZMAPA	requires the map database be opened on Fortran logical unit 1.
PLOTCHAR	requires the font database be opened on Fortran logical unit 3.
PWRITX	requires the font database be opened on Fortran logical unit 3.
ISOSRFHR	requires a scratch file be opened on unit IUNIT, which is passed via COMMON/UNITS/IUNIT.

The ultimate test for success or failure is a comparison of the output plots with the plots that are listed in the table on the next page.

Where to Find the Demonstration Plots for Each Utility

Test File	Utility	Plot Location*
TAREAS	AREAS	Version 2.00, Examples section
TAUTOG	AUTOGRAPH	Version 2.00, Examples section
TCOLCV	COLCONV	No plotted output
TCONPA	CONPACK	Version 3.00, pp. 3-151 through 3-153
TCONAN	CONRAN	Version 2.00, Examples section
TCONAQ	CONRAQ	Version 2.00, Examples section
TCONAS	CONRAS	Version 2.00, Examples section
TCONRE	CONREC	Version 2.00, Examples section
TCNQCK	CONRECQCK	Version 2.00, Examples section
TCNSMT	CONRECSMTH	Version 2.00, Examples section
TCNSUP	CONRECSPR	Version 2.00, Examples section
TDASHC	DASHCHAR	Version 2.00, Examples section
TDASHL	DASHLINE	Version 2.00, Examples section
TDASHS	DASHSMTH	Version 2.00, Examples section
TDASHP	DASHSUPR	Version 2.00, Examples section
TEZMAP	EZMAP	Version 2.00, Examples section
TEZMPA	EZMAPA	Version 2.00, Examples section
TGFLAS	GFLASH	Version 3.00, pp. 6-13 through 6-15
TGRIDA	GRIDAL	Version 2.00, Examples section
THAFTO	HAFTON	Version 2.00, Examples section
THSTGR	HISTGR	Version 2.00, Examples section
TISOSR	ISOSRF	Version 2.00, Examples section
TISOHR	ISOSRFHR	Version 2.00, Examples section
TLBLBA	LABELBAR	Version 3.00, p. 5-63
TPLTCH	PLOTCHAR	Version 3.00, pp. 5-31 through 5-40
TPWRTX	PWIRTX	Version 2.00, Examples section
TPWRTY	PWRITY	Version 2.00, Examples section
TPWRZI	PWRZI	Version 2.00, Examples section
TPWRZS	PWRZS	Version 2.00, Examples section
TPWRZT	PWRZT	Version 2.00, Examples section
TSOFTF	SOFTFILL	Version 3.00, pp. 4-19 through 4-23
TSRFAC	SRFACE	Version 2.00, Examples section
TSTITL	STITLE	Version 3.00, p. 6-47
TSTRML	STRMLN	Version 2.00, Examples section
TTHREE	THREED	Version 2.00, Examples section
TVELVC	VELVCT	Version 2.00, Examples section

*Version 2.00 = *NCAR Graphics User's Guide, Version 2.00*
*Version 3.00 = *NCAR Graphics Guide to New Utilities, Version 3.00*

To create the executable module for any given demo driver, use the following source files:

- A main program that invokes the driver subroutine,
- the driver subroutine for the given utility as supplied on the distribution tape,
- the source for the higher-level utility,
- SPPS,
- a GKS package at level at least 0A, and
- the low-level support routines.

When implementing any higher-level utility, be sure to read over the documentation for that utility as supplied in the two User's Guides for any special implementation considerations. For example:

- EZMAP, PWRITX, and PLOTCHAR require connecting to support data files. The support data file for EZMAP is contained in EZMAPDAT on the distribution tape; the support data files for PWRITX and PLOTCHAR are contained in files PWRITXC1, PWRITXC2, PWRITXD1, and PWRITXD2 on the distribution tape. See the documentation in the code for EZMAP for instructions on how to process the ASCII file EZMAPDAT to create the binary file required by EZMAP. Consult the documentation in the file PWRITXNT for creating the binary database for PWRITX and PLOTCHAR. As noted above, PLOTCHAR supersedes PWRITX and PWRITY and has a much simpler interface, it is recommended that PLOTCHAR be implemented and not PWRITX and PWRITY. PWRITX and PWRITY are made available on Version 3.00 for compatibility with previous versions of the package.
- ISOSRFHR requires a scratch file.
- DASHSUPR requires some code modification. See the code for DASHSUPR for details on its implementation.
- If DASHSMTH is used with CONREC, then the contour lines are passed through a smoother.

An installer should be aware of the possibility of naming conflicts between entries in NCAR Graphics and names already on the local system. There are several entry point names in NCAR Graphics which are particularly subject to duplication. A list of these names is:

ANOTAT	BKGND	BOUND	CURVE	CURVE3	CURVED
DISPLA	DRAWS	DRAWT	FENCE3	FILLIN	FRAME
FRSTPT	GAP	GETSET	GRAY	GRID	GTNUM
GTNUMB	GTSIGN	IDIOT	INIT3D	LINE	LINE3
LINED	MAPIT	MAPIO	MAPSAV	MAPSET	MINMAX
PERIM	PERIM3	PLOTIT	POINT	POINT3	POINTS
REORD	RESET	SET	SET3	SET3D	SETER
SETI	SETR	TICK3	TICKS	VECT3	VECTOR

Additional Examples Available

In addition to the demo drivers for the higher-level utilities, the distribution tape also contains more extensive examples of some of the utilities. Besides serving as examples, in many cases these files serve the additional function of providing a template for user modifications to meet specific user requirements. Most of the examples are stand alone programs. The files containing the additional examples are:

Additional Examples Files

File	Contents
EAUTOG	Contains 13 examples for AUTOGRAPH. The final example requires a data file that is also supplied.
ECONPA	Contains 9 examples for CONPACK. These examples all require some common code that is also supplied.
EEZMAP	Contains 10 examples for EZMAP. The final example requires a data file that is also supplied.
EEZMPA	Contains one additional example for EZMAPA.
ELBLBA	Contains one additional example for ELBLBA.
EMISC	Contains five miscellaneous codes. There are codes for the color charts, for the sixteen sample colors, and for the HSV color wheels. The code indicating how to use BIVAR with CONPACK to handle randomly spaced data is included as well as a supplementary example for STITLE.
EPLTCH	Contains one additional example for PLOTCHAR.
ESOFTF	Contains two examples for SOFTFILL.

In the files that have multiple examples, each individual example code is separated with two special separator lines beginning with a line of the form

NCAR_FILE_START name

and ending with a line of the form

NCAR_FILE_END name

The *name* in the above lines is a name for the example code which is contained between the separator lines. The two special separator lines must be stripped off before using the Fortran code contained. It is an easy task to write a procedure that splits the master example files into the desired individual examples.

User Libraries

How the files are organized and presented to the end user is pretty much up to the implementor. The following components of the package should be made available in some form (as binary libraries, or source, or however):

1. **A GKS package, level 0A at minimum.** This may be a commercial product, or it may be the GKS package written by NCAR. If GFLASH or STITLE will be used at your site, you must use either the NCAR GKS package (which contains some needed level 2A enhancements) or a commercial package at level 2A or higher.
2. **SPPS, the System Plot Package Simulator.**
3. **Higher-level utilities.** These are the primary user interface to the package. Not all of the utilities can be put into the same binary library because of duplication of entry point names in the contouring family (CONRCQCK, CONREC, CONRCSPR) and the DASH family (DASHLINE, DASHCHAR, DASHSMTH, DASHSUPR). Since CONPACK replaces the old contouring family and CONPACK does not have duplicate entry point names, use it in your library unless you need the old contouring package for compatibility with previous versions of the package.
4. **User aids.** The routine FINDG should be provided if users will be wanting to convert their applications from using a pre-GKS version of the NCAR package to the current GKS version.
5. **Demo drivers and examples.** Users find that it is very helpful to take the source for the demo drivers and examples and modify them to suit their own particular needs.
6. **NCAR Computer Graphics Metafile translator.**

Size Considerations

It is not possible to give precise data on the memory requirements for installation of the package, since different systems will provide different libraries, different linkers, and so on. We can give a typical example. On a Sun 3/260 computer, the object module created for running the CONPACK demonstration driver (sources; application driver; the CONPACK demo TCONPA; the CONPACK and required subsidiary utilities; SPPS; NCAR's GKS package; and the low-level support routines) is 1,171,456 bytes in size. Different utilities will of course have different space requirements. The binary database used by EZMAP requires 587,264 bytes on a Sun, for example.

To provide an estimate of the size of metafiles created by the NCAR GKS package, the metafile sizes, in bytes, for all of the demonstration driver plots for all of the higher-level utilities are listed in the following table.

Metafile Sizes

Utility	CGM Size
AREAS	15840
AUTOGRPH	18720
CONPACK	47520
CONRAN	15840
CONRAQ	24480
CONRAS	21600
CONRCQCK	17280
CONRCSPR	24480
CONREC	18720
DASHCHAR	7200
DASHLINE	7200
DASHSMTH	8640
DASHSUPR	10080
EZMAP	1344960
EZMAPA	24480
GFLASH	41760
GRIDAL	25920
HAFTON	491040
HISTGR	11520
ISOSRF	38880
ISOSRFHR	38880
LABELBAR	50400
PLOTCHAR	396000
PWRITX	95040
PWRITY	8640
PWRZI	20160
PWRZS	17280
PWRZT	5760
SOFTFILL	46080
SRFACE	21600
STITLE	10080
STRMLN	14400
THREED	8640
VELVCT	31680

Appendix A: Machine-dependent Support Routines

Following are functional descriptions of the required locally-implemented support routines. A test suite is distributed for this package so that an implementor may verify that the implementations are correct. The routine G01MIO is needed only if files AWI and BWI (NCAR's GKS package) are being implemented.

FUNCTION I1MACH(I)

This function is used to set up 16 machine constants:

I1MACH(1) = the standard input unit
I1MACH(2) = the standard output unit
I1MACH(3) = the standard punch unit
I1MACH(4) = the standard error message unit
I1MACH(5) = the number of bits per integer storage unit
I1MACH(6) = the number of characters per integer storage unit

Assume that integers are represented in the S-digit, base-A form:

SIGN*(X(S-1)*A**(S-1)+...+X(1)*A+X(0))

in which 0 .LT. X(I) .LT. A for I=0,...,S-1.

I1MACH(7) = A, the base
I1MACH(8) = S, the number of base-A digits
I1MACH(9) = A**S-1, the largest magnitude

Assume that floating-point numbers are represented in the T-digit, base-B form:

SIGN*(B**E)*((X(1)/B+...+(X(T)/B**T))

in which 0 .LT. X(1), and EMIN .LE. E .LE. EMAX.

I1MACH(10) = B, the base

Single-precision Constants
1MACH(11) = T, the number of base-B digits
I1MACH(12) = EMIN, the smallest exponent E
I1MACH(13) = EMAX, the largest exponent E

Double-precision Constants
I1MACH(14) = T, the number of base-B digits
I1MACH(15) = EMIN, the smallest exponent E
I1MACH(16) = EMAX, the largest exponent E

FUNCTION R1MACH(I)

This function sets 5 single-precision machine constants:

R1MACH(1) = B**(EMIN-1), the smallest positive magnitude
R1MACH(2) = B**EMAX*(1-B**(-T)), the largest magnitude
R1MACH(3) = B**(-T), the smallest relative spacing
R1MACH(4) = B**(1-T), the largest relative spacing
R1MACH(5) = LOG10(B)

FUNCTION ISHIFT(IWORD,N)

IWORD is shifted by N bits. If $N > 0$, a left circular shift is performed (all bits are shifted left N bits, and the bits that are shifted out of the word to the left are shifted back into the word at the right). If $N < 0$, a right end-off shift is performed (all bits are shifted right by N bits, and the bits that are shifted out of the right of the word are lost) — if the left-most bit is 0, then the vacated positions are filled with zeros; if the leftmost bit is 1, then the vacated positions are undefined.

The implementor may assume that IABS(N) .LE. (word length).

FUNCTION IAND(K1,K2)

The bit-by-bit logical product of K1 and K2. If K3 = IAND(K1,K2), then the i-th bit of K3 is 0 if the i-th bit of either K1 or K2 is 0; otherwise the i-th bit of K3 is 1.

FUNCTION IOR(K1,K2)

The bit-by-bit logical sum of K1 and K2. If K3 = IOR(K1,K2), then the i-th bit of K3 is 0 if and only if the i-th bit of both K1 and K2 is 0.

SUBROUTINE G01MIO (IOP, IUNIT, FNAME, IBUFF, LENGTH, IERROR)

This output routine is the central one for the metafile generator. A Fortran implementation of this subroutine is distributed as part of the BWI file. The Fortran implementation of G01MIO contained in BWI is correct FORTRAN 77.

Input Parameters

IOP — Indicates type of operation desired:
IOP = 1, OPEN workstation for output on IABS(IUNIT). IOP = 2, CLOSE workstation for output on IABS(IUNIT). IOP = 3, write IBUF to IABS(IUNIT). IOP = 4, read IABS(IUNIT) into IBUF. IOP = 5, position the record pointer to the beginning of the file. IOP = 6, position the record pointer to the beginning of the previous record.

FNAME — The filename to be used for the open operation.

IUNIT — IABS(IUNIT) is the Fortran logical unit number on which IOP is to occur.

IBUFF — Buffer containing data for a read or write operation.

LENGTH — Length of data in IBUFF, in integer words.

Output Parameters

IERROR — Error indicator that equals 0 if no errors.

SUBROUTINE GBYTES(NPACK,ISAM,IBIT,NBITS,NSKIP,ITER)

This subroutine is used to unpack bit chunks from NPACK into the ISAM array. A portable Fortran version of this routine is distributed, but the Fortran version is inefficient and should be replaced with a more efficient implementation.

NPACK — Address of first word of array to be unpacked. For the purposes of this subroutine, NPACK is viewed as a bit stream.

ISAM — Array to receive the unpacked bit chunks. They will be right-justified with zero-fill in this array. ISAM should be dimensioned for ITER.

IBIT — A bit-count offset to be used before the first bit chunk is unpacked. For example, if IBIT=3, and NBITS=5, then 3 bits in NPACK will be skipped and the next 5 bits will be unpacked into ISAM(1).

NBITS — The number of bits in each bit chunk to be unpacked. An error condition occurs if NBITS is larger than the number of bits-per-word on the given machine.

NSKIP — The number of bits to skip between each bit chunk to be unpacked. Bits are skipped only after the first bit chunk has been unpacked.

ITER — The number of bit chunks to be unpacked. For example:

```
CALL GBYTES(NPB,ISB,3,6,9,2)
```

In this call, 3 bits would be skipped at the beginning of NPB; the next 6 bits would be unpacked into ISB(1) and right-justified with zero-fill; 9 bits would be skipped in NPB, and then the next six bits of NPB would be unpacked into ISB(2) and right-justified with zero-fill.

SUBROUTINE SBYTES(NPACK,ISAM,IBIT,NBITS,NSKIP,ITER)

This subroutine is the reverse of GBYTES as described above.

NPACK — Address of first word of array to be packed.

ISAM — Array to be packed into NPACK. The right-most NBITS bits of each word will be packed. ISAM should be dimensioned for at least ITER.

IBIT — A bit-count offset to be used before the first bits are packed into NPACK. For example, if IBIT=3, and NBITS=5, 3 bits in NPACK will be skipped before the right-most 5 bits of ISAM(1) are packed into it.

NBITS — The number of bits in each word of ISAM to be unpacked. An error condition occurs if NBITS exceeds the word size on the given machine.

NSKIP — The number of bits to skip between each bit chunk packed.

ITER — The number of bit chunks to be packed. For example:

```
CALL SBYTES(NPC,ISB,45,6,3,2)
```

In this call, 45 bits would be skipped at the beginning of NPC; the right-most 6 bits of ISB(1) would be packed into NPC; 3 bits would be skipped in NPC, and the right-most 6 bits of ISB(2) would be packed into NPC.

Appendix B: Dumps of the Metafile from TGKS0A

Following are octal and hex dumps of the metafile produced from test driver GKS0A. The metafile has three 1440 byte records.

Octal Dump — Record #1

Decimal Address	Bytes dumped in octal															
	Record #1															
0001	002	045	064	000	000	077	000	075	074	040	040	040	040	040	040	040
0017	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040
0033	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040
0049	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040
0065	040	040	040	040	040	000	020	042	000	001	020	131	030	116	103	101
0081	122	137	107	113	123	060	101	055	055	126	105	122	123	111	117	116
0097	137	063	056	060	060	000	021	146	000	001	377	377	000	000	021	277
0113	001	267	007	104	105	106	101	125	114	124	032	110	105	122	123	110
0129	105	131	072	103	101	122	124	117	107	122	101	120	110	111	103	137
0145	122	117	115	101	116	032	110	105	122	123	110	105	131	072	103	101
0161	122	124	117	107	122	101	120	110	111	103	137	107	122	105	105	113
0177	025	110	105	122	123	110	105	131	072	123	111	115	120	114	105	130
0193	137	122	117	115	101	116	025	110	105	122	123	110	105	131	072	123
0209	111	115	120	114	105	130	137	107	122	105	105	113	026	110	105	122
0225	123	110	105	131	072	123	111	115	120	114	105	130	137	123	103	122
0241	111	120	124	025	110	105	122	123	110	105	131	072	103	117	115	120
0257	114	105	130	137	122	117	115	101	116	025	110	105	122	123	110	105
0273	131	072	103	117	115	120	114	105	130	137	107	122	105	105	113	026
0289	110	105	122	123	110	105	131	072	103	117	115	120	114	105	130	137
0305	123	103	122	111	120	124	026	110	105	122	123	110	105	131	072	103
0321	117	115	120	114	105	130	137	111	124	101	114	111	103	030	110	105
0337	122	123	110	105	131	072	103	117	115	120	114	105	130	137	103	131
0353	122	111	114	114	111	103	024	110	105	122	123	110	105	131	072	104
0369	125	120	114	105	130	137	122	117	115	101	116	025	110	105	122	123
0385	110	105	131	072	124	122	111	120	114	105	130	137	122	117	115	101
0401	116	026	110	105	122	123	110	105	131	072	124	122	111	120	114	105
0417	130	137	111	124	101	114	111	103	025	110	105	122	123	110	105	131
0433	072	107	117	124	110	111	103	137	107	105	122	115	101	116	026	110
0449	105	122	123	110	105	131	072	107	117	124	110	111	103	137	105	116
0465	107	114	111	123	110	026	110	105	122	123	110	105	131	072	107	117
0481	124	110	111	103	137	111	124	101	114	111	101	116	024	110	105	122
0497	123	110	105	131	072	115	101	124	110	137	123	131	115	102	117	114
0513	123	023	110	105	122	123	110	105	131	072	123	131	115	102	117	114
0529	137	123	105	124	061	023	110	105	122	123	110	105	131	072	123	131
0545	115	102	117	114	137	123	105	124	062	000	000	000	000	000	000	000
0561	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000
	The remaining 864 bytes of record #1 are zero															

Octal Dump — Record #2

Decimal Address	Bytes dumped in octal															
	Record #2 (1440 bytes total)															
0001	002	166	070	000	000	177	000	075	074	040	040	040	040	040	040	040
0017	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040
0033	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040
0049	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040	040
0065	040	040	040	040	040	000	000	200	124	104	000	000	000	000	124	104
0081	001	314	314	314	060	302	000	001	060	250	014	314	014	314	163	062
0097	163	062	100	077	000	044	021	353	141	106	027	012	153	204	034	050
0113	141	106	041	107	153	204	046	146	141	106	053	204	153	204	060	243
0129	141	106	065	302	153	204	072	340	141	106	120	302	000	002	100	177
0145	000	044	021	353	107	255	027	012	121	352	034	050	107	255	041	107
0161	121	352	046	146	107	255	053	204	121	352	060	243	107	255	065	302
0177	121	352	072	340	107	255	122	302	000	001	124	050	000	000	105	036
0193	105	036	000	000	123	344	077	377	077	377	100	377	000	044	137	174
0209	153	117	132	355	155	371	125	301	155	015	122	140	151	002	122	140
0225	143	311	125	301	137	275	132	355	136	321	137	174	141	173	141	106
0241	146	145	101	077	001	064	112	074	107	255	150	364	124	172	150	364
0257	107	255	000	030	000	014	000	000	000	001	001	000	001	000	001	000
0273	001	000	001	000	001	000	001	000	001	000	001	000	001	000	001	000
0289	001	000	000	001	000	001	000	001	000	001	000	001	000	001	000	001
0305	000	001	000	001	000	001	000	001	000	001	001	000	001	000	001	000
0321	001	000	001	000	001	000	001	000	001	000	001	000	001	000	001	000
0337	001	000	000	001	000	001	000	001	000	001	000	001	000	001	000	001
0353	000	001	000	001	000	001	000	001	000	001	001	000	001	000	001	000
0369	001	000	001	000	001	000	001	000	001	000	001	000	001	000	001	000
0385	001	000	000	001	000	001	000	001	000	001	000	001	000	001	000	001
0401	000	001	000	001	000	001	000	001	000	001	001	000	001	000	001	000
0417	001	000	001	000	001	000	001	000	001	000	001	000	001	000	001	000
0433	001	000	000	001	000	001	000	001	000	001	000	001	000	001	000	001
0449	000	001	000	001	000	001	000	001	000	001	001	000	001	000	001	000
0465	001	000	001	000	001	000	001	000	001	000	001	000	001	000	001	000
0481	001	000	000	001	000	001	000	001	000	001	000	001	000	001	000	001
0497	000	001	000	001	000	001	000	001	000	001	001	000	001	000	001	000
0513	001	000	001	000	001	000	001	000	001	000	001	000	001	000	001	000
0529	001	000	000	001	000	001	000	001	000	001	000	001	000	001	000	001
0545	000	001	000	001	000	001	000	001	000	001	122	114	000	002	000	003
0561	000	000	000	000	000	000	000	000	121	342	002	014	122	010	000	000
0577	002	014	002	014	000	000	100	237	000	055	077	377	046	146	000	000
0593	046	105	170	141	155	160	154	145	040	163	164	162	151	156	147	054
0609	040	143	145	156	164	145	162	145	144	040	151	156	040	164	150	145
0625	040	143	145	156	164	145	162	000	000	240	000	000	000	000	000	000
0641	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000	000
	The remaining 784 bytes of record #2 are zero															

Octal Dump — Record #3

Decimal Address	Bytes dumped in octal
	Record #3 (1440 bytes total) (Only the first 6 bytes of this record are significant, the other bytes may be non-zero, but are of no consequence.)
0001	000 002 062 000 000 100 ... ...

Hex Dump — Record #1

Decimal Address	Bytes dumped in hex							
	Record #1 (1440 bytes total)							
0001	0225	3400	003F	003D	3C20	2020	2020	2020
0017	2020	2020	2020	2020	2020	2020	2020	2020
0033	2020	2020	2020	2020	2020	2020	2020	2020
0049	2020	2020	2020	2020	2020	2020	2020	2020
0065	2020	2020	2000	1022	0001	1059	184E	4341
0081	525F	474B	5330	412D	2D56	4552	5349	4F4E
0097	5F33	2E30	3000	1166	0001	FFFF	0000	11BF
0113	01B7	0744	4546	4155	4C54	1A48	4552	5348
0129	4559	3A43	4152	544F	4752	4150	4849	435F
0145	524F	4D41	4E1A	4845	5253	4845	593A	4341
0161	5254	4F47	5241	5048	4943	5F47	5245	454B
0177	1548	4552	5348	4559	3A53	494D	504C	4558
0193	5F52	4F4D	414E	1548	4552	5348	4559	3A53
0209	494D	504C	4558	5F47	5245	454B	1648	4552
0225	5348	4559	3A53	494D	504C	4558	5F53	4352
0241	4950	5415	4845	5253	4845	593A	434F	4D50
0257	4C45	585F	524F	4D41	4E15	4845	5253	4845
0273	593A	434F	4D50	4C45	585F	4752	4545	4B16
0289	4845	5253	4845	593A	434F	4D50	4C45	585F
0305	5343	5249	5054	1648	4552	5348	4559	3A43
0321	4F4D	504C	4558	5F49	5441	4C49	4318	4845
0337	5253	4845	593A	434F	4D50	4C45	585F	4359
0353	5249	4C4C	4943	1448	4552	5348	4559	3A44
0369	5550	4C45	585F	524F	4D41	4E15	4845	5253
0385	4845	593A	5452	4950	4C45	585F	524F	4D41
0401	4E16	4845	5253	4845	593A	5452	4950	4C45
0417	585F	4954	414C	4943	1548	4552	5348	4559
0433	3A47	4F54	4849	435F	4745	524D	414E	1648
0449	4552	5348	4559	3A47	4F54	4849	435F	454E
0465	474C	4953	4816	4845	5253	4845	593A	474F
0481	5448	4943	5F49	5441	4C49	414E	1448	4552
0497	5348	4559	3A4D	4154	485F	5359	4D42	4F4C
0513	5313	4845	5253	4845	593A	5359	4D42	4F4C
0529	5F53	4554	3113	4845	5253	4845	593A	5359
0545	4D42	4F4C	5F53	4554	3200	0000	0000	0000
0561	0000	0000	0000	0000	0000	0000	0000	0000
	The remaining 864 bytes of record #1 are zero							

Hex Dump — Record #2

Decimal Address	Bytes dumped in hex							
	Record #2 (1440 bytes total)							
0001	0276	3800	007F	003D	3C20	2020	2020	2020
0017	2020	2020	2020	2020	2020	2020	2020	2020
0033	2020	2020	2020	2020	2020	2020	2020	2020
0049	2020	2020	2020	2020	2020	2020	2020	2020
0065	2020	2020	2000	0080	5444	0000	0000	5444
0081	01CC	CCCC	30C2	0001	30A8	0CCC	0CCC	7332
0097	7332	403F	0024	11EB	6146	170A	6B84	1C28
0113	6146	2147	6B84	2666	6146	2B84	6B84	30A3
0129	6146	35C2	6B84	3AE0	6146	50C2	0002	407F
0145	0024	11EB	47AD	170A	51EA	1C28	47AD	2147
0161	51EA	2666	47AD	2B84	51EA	30A3	47AD	35C2
0177	51EA	3AE0	47AD	52C2	0001	5428	0000	451E
0193	451E	0000	53E4	3FFF	3FFF	40FF	0024	5F7C
0209	6B4F	5AED	6DF9	55C1	6D0D	5260	6902	5260
0225	63C9	55C1	5FBD	5AED	5ED1	5F7C	617B	6146
0241	6665	413F	0134	4A3C	47AD	68F4	547A	68F4
0257	47AD	0018	000C	0000	0001	0100	0100	0100
0273	0100	0100	0100	0100	0100	0100	0100	0100
0289	0100	0001	0001	0001	0001	0001	0001	0001
0305	0001	0001	0001	0001	0001	0100	0100	0100
0321	0100	0100	0100	0100	0100	0100	0100	0100
0337	0100	0001	0001	0001	0001	0001	0001	0001
0353	0001	0001	0001	0001	0001	0100	0100	0100
0369	0100	0100	0100	0100	0100	0100	0100	0100
0385	0100	0001	0001	0001	0001	0001	0001	0001
0401	0001	0001	0001	0001	0001	0100	0100	0100
0417	0100	0100	0100	0100	0100	0100	0100	0100
0433	0100	0001	0001	0001	0001	0001	0001	0001
0449	0001	0001	0001	0001	0001	0100	0100	0100
0465	0100	0100	0100	0100	0100	0100	0100	0100
0481	0100	0001	0001	0001	0001	0001	0001	0001
0497	0001	0001	0001	0001	0001	0100	0100	0100
0513	0100	0100	0100	0100	0100	0100	0100	0100
0529	0100	0001	0001	0001	0001	0001	0001	0001
0545	0001	0001	0001	0001	0001	524C	0002	0003
0561	0000	0000	0000	0000	51E2	020C	5208	0000
0577	020C	020C	0000	409F	002D	3FFF	2666	0000
0593	2645	7861	6D70	6C65	2073	7472	696E	672C
0609	2063	656E	7465	7265	6420	696E	2074	6865
0625	2063	656E	7465	7200	00A0	0000	0000	0000
0641	0000	0000	0000	0000	0000	0000	0000	0000
	The remaining 864 bytes of record #2 are zero							

Hex Dump — Record #3

Decimal Address	Bytes dumped in hex
	Record #3 (1440 bytes total) (Only the first 6 bytes of this record are significant, the other bytes may be non-zero, but are of no consequence.)
0001	0002 3200 0040 ... ...

Section III: Release Contents

Section III: Release Contents

Release Contents

This section gives brief descriptions for all files on the distribution tape. The files are separated into logical groupings. In the list that follows, the titles beginning on the left margin indicate the groupings (for example, "Required Support Routines," below); the titles at the first level of indentation correspond to the filenames on the distribution tape (for example, "SUPPORT," below); the boldface titles at the second level of indentation correspond to routines of interest in the files (for example, "ENCD," below).

Unless otherwise noted, assume that all software in the NCAR Graphics package is written in FORTRAN 77 and adheres strictly to that standard.

Included in the description of a file is a list of other files needed to properly assemble and run that file. However, the routines in the files LOCAL, SUPPORT, and SPPS are used throughout the package, and thus it is assumed that they will be available to every routine.

The two files in the Bookkeeping group below are the first two files on the tape. They contain information on how the files are ordered on the tape and when the tape was created.

Bookkeeping

FNAMES This file contains the names of all files on the distribution tape in the sequence in which they appear on the tape. Each name is on a separate 80-character line and is in uppercase ASCII with blank fill.

HEADER This is a two-line file identifying the version of the package and the date and time the original distribution tape was created.

NCAR GKS 0A

AWI Standing for "Above Workstation Interface," AWI codes up the parameters from the various GKS calls into a single common block and then, when appropriate, invokes BWI (which follows below).

BWI Standing for "Below Workstation Interface," upon being invoked by AWI, BWI takes the coded parameters in a single common block and generates the appropriate CGM instructions.

Required Support Routines

SUPPORT This file contains a collection of FORTRAN 77 routines required by code throughout the package. It should be implemented in all cases.

ENCD: Used by the VELVCT and CONREC utilities to generate ASCII labels from numeric values by encoding them in an appropriate format.

ERPRT77: An error handling package, adapted from the non-proprietary part of the PORT Mathematical Subroutine Library from Bell Labs.

Q8QST4: This routine is distributed as a dummy. It is called from most user entry points, and it may be used for gathering usage statistics.

MSKRV1, MSKRV2, MSBSF1, MSBSF2: One-dimensional and two-dimensional smoothers.

LOCAL

This file contains examples of the lowest level machine-dependent subroutines that need to be implemented on each machine. Fortran and/or C versions of these routines are provided as samples of what should be done, but it is recommended that they be implemented in assembly language since they are used extensively by the entire package. It is important to realize that the supplied routines are **examples only** and are not necessarily meant to run on your machine. If you try to use them on your machine, care must be taken to make sure that the functional descriptions in Appendix A of the "NCAR Graphics Installation Directions" section of this manual are satisfied.

IOR: A function that results in the bit-by-bit logical sum of its two arguments.

IAND: A function that results in the bit-by-bit logical product of its two arguments.

ISHIFT: A function that, based on the integer value of its second argument, N, will perform on its first argument a left circular shift by N bits if N is positive, or right end-off shift by N bits if N is negative.

GBYTES: Unpacks bits from a bit-string given in an array, and puts them into another array.

SBYTES: Packs bits from an array into another sequential array.

I1MACH: A function that returns 1 of 16 machine-dependent integer-valued constants.

R1MACH: A function that returns 1 of 5 machine-dependent real-valued constants.

Implementation Tools

IMPLTEST

TGKS0A: A program that generates an example of five GKS output primitives.

TLOCAL: A program that tests correctness of implementation for the required local routines IAND, IOR, ISHIFT, GBYTES, and SBYTES.

TPLOT: A program that produces a single simple plot that depends only on GKS and SPPS. The code contains octal and hex dumps of the metafile that should be produced.

TSPPS: A program that extensively tests all SPPS (System Plot Package Simulator) entries.

FINDG A program for locating all calls to entries of pre-GKS NCAR Graphics, which may require modification to run in the GKS package.

Utilities

All the higher-level utilities files are listed here. However, with the release of Version 3.00, several of these utilities are superseded by new utilities in Version 3.00. The new utilities duplicate the functionality of certain older utilities, and, in different utilities, offer additional functionality, color capabilities, or easier-to-use interfaces. Therefore, we urge you to urge your users to use the new utilities whenever possible. The files for the "replaced" utilities are provided here for backward-compatibility only.

BIVAR, public domain software that interpolates from random data to gridded data, is included in the Version 3.00 release. We suggest that users use BIVAR in combination with CONPACK to produce contours from random data. The BIVAR/CONPACK combination supersedes the use of CONRAN, CONRAQ, and CONRAS, three utilities that are known to contain elusive bugs. The file EMISC contains BIVAR as well as an example showing how to use BIVAR with CONPACK. It is documented internally in the file.

AREAS Creates an area map from a set of edges that divide a two-dimensional plane into areas. The area map may then be used in a number of ways, such as in conjunction with EZMAPA to create solid-colored maps.

AUTOGRPH Draws and annotates curves or families of curves. Requires DASHCHAR (or DASHSMTH, if smoothed curves are desired). This utility is also referred to as AUTOGRAPH.

COLCONV Converts color specifications given in one color space to another. Options include Hue, Lightness, and Saturation (HLS), Red, Green, Blue (RGB), Hue, Saturation, Value (HSV), and the YIQ space. New in Version 3.00.

CONPACK Draws black-and-white or color contour plots from regularly spaced (gridded) data. CONPACK allows for easy mapping of output, particularly onto an EZMAP background. New in Version 3.00. Supersedes and upgrades CONREC, CONRCQCK, and CONRCSPR. In combination with BIVAR, also supersedes CONRAN, CONRAQ, and CONRAS.

CONRAN Contours irregularly spaced data, labeling the contour lines. Requires DASHCHAR (or DASHSMTH, if smoothed curves are desired). Also requires CONCOM and CONTERP. Superseded by the combination of BIVAR and CONPACK.

CONRAQ Like CONRAN, but smaller and faster because it has no labeling capacity. Also requires CONTERP. Superseded by the combination of BIVAR and CONPACK.

CONRAS Like CONRAN, but bigger and slower because lines are smoothed and crowded lines are removed. Also requires CONCOM and CONTERP. Superseded by the combination of BIVAR and CONPACK.

CONREC Contours two-dimensional arrays, labeling the contour lines. Requires DASHCHAR (or DASHSMTH, if smoothed contour lines are desired). Superseded by CONPACK.

CONRCQCK Like CONREC, but faster and smaller because contours are unlabeled. This package shares entry names with CONREC, so they cannot both be included in a binary library. This utility is also referred to as CONRECQCK. Superseded by CONPACK.

CONRCSPR Like CONREC, but bigger and slower because contours are smoothed and labeled, and crowded lines are removed. This package shares entry names with CONREC, so they cannot both be included in a binary library. This utility is also referred to as CONRECSPR. Superseded by CONPACK.

DASHCHAR Provides a dashed line package with labeling capability.

DASHLINE Like DASHCHAR, but smaller and faster because it has no labeling capacity. This package shares entry names with DASHCHAR, so they cannot both be included in a binary library.

DASHSMTH Like DASHCHAR, but bigger and slower because lines are smoothed. This package shares entry names with DASHCHAR, so they cannot both be included in a binary library.

DASHSUPR Like DASHCHAR, but bigger and slower because lines are smoothed and crowded lines are removed. This package shares entry names with DASHCHAR, so they cannot both be included in a binary library.

EZMAP Plots continental, U.S. state, and world political outlines according to one of ten projections. Requires the continental outline database EZMAPDAT.

EZMAPA Allows EZMAP output to be redirected to routines in the AREAS package. EZMAPA makes it possible to create solid-colored world maps and to draw lines on a map, masked by the areas created by the area map (lines of latitude and longitude omitted over land masses, for example).

GFLASH Provides for a limited picture segmentation capability. Parts of pictures can be saved in buffers and inserted into any subsequent picture. GFLASH duplicates the functionality of the FLASH package in the pre-GKS version of NCAR Graphics. New in Version 3.00.

GRIDAL Provides for drawing graph paper, backgrounds, perimeters, and so on.

HAFTON Creates halftone (gray scale) pictures from a two-dimensional array.

HISTGR Provides a general purpose package for drawing histograms (bar charts).

ISOSRF Creates iso-surfaces (with hidden lines removed) from a three-dimensional array. You may optionally include PWRZI for labeling the plot.

ISOSRFHR Creates iso-surfaces (with hidden lines removed) from a high resolution three-dimensional array.

LABELBAR Draws a rectangular bar that may be filled, using color or patterns, and labeled to serve as a key for a filled plot. New in Version 3.00.

PLOTCHAR Provides for producing characters at various quality levels. PLOTCHAR supersedes PWRITX and PWRITY and provides a much simpler interface. The database files PWRITXC1, PWRITXC2, PWRITXD1, and PWRITXD2 are required as for PWRITX, see below. New in Version 3.00.

PWRITX Draws characters using the Hershey database. Requires that the database files PWRITXC1, PWRITXC2, PWRITXD1, and PWRITXD2 be run first through the program PWRITXNT to turn card-image fonts into binary form. Superseded by PLOTCHAR.

PWRITY Draws simple software characters. Superseded by PLOTCHAR.

PWRZI Plots characters in three-dimensional space. It is used only with the utility ISOSRF.

PWRZS Plots characters in three-dimensional space. It is used only with the utility SRFACE.

PWRZT Plots characters in three-dimensional space. It is used only with the utility THREED.

SOFTFILL Fills polygons in software using parallel solid lines, rows of dots, polymarkers, or rows of characters. Can do cross-hatching, dot and line patterns may be specified, and color may be used. New in Version 3.00.

SRFACE Creates a three-dimensional display of a function of two variables (with hidden lines removed). You may optionally use PWRZS for labeling the plot.

STITLE Produces scrolled or stationary movie titles. Color is available as well as fade in and fade out. This package replaces and augments the SCROLL package in the pre-GKS version of NCAR Graphics. New in Version 3.00.

STRMLN Plots a representation of any two-dimensional vector field for which planar vector components are given on a regular rectangular lattice, displaying both field direction (via lines of flow containing arrowheads and feathers) and field magnitude (based on distance between those flow lines).

THREED Provides three-dimensional line-drawing capabilities, with entry points equivalent to the line drawing entry points of the System Plot Package Simulator. You may optionally use PWRZT for labeing the plots.

VELVCT Creates two-dimensional velocity field displayed by drawing arrows from the data locations.

Utility Support Files

AGUPWRTX A version of the AGPWRT subroutine of the AUTOGRAPH package. AGUPWRTX allows AUTOGRPH to use the character set from the PWRITX utility.

BNCHMK A program that generates six complex examples of usage of the higher-level utilities. Particularly examples are provided for overlaying the output from one higher-level utility onto another. It calls the following utilities: GRIDAL, AUTOGRPH, CONREC, DASHCHAR, VELVCT, and EZMAP. Additionally, it requires the database file RANFDAT, which is

	a separate file on the tape. This program requires internal modifications before use; please read the documentation in the code for instructions.
CONCOM	A set of routines used by the CONRAN and CONRAS packages.
CONTERP	A set of routines used by the CONRAN, CONRAQ, and CONRAS packages.
SPPS	Standing for the "System Plot Package Simulator," this collection of routines simulates the pre-GKS NCAR System Plot Package by containing the same entry points, which in turn call the appropriate GKS routines to accomplish the given task. These routines are used extensively by the higher-level utilities. For users of the pre-GKS package, these routines should prove helpful in the conversion to the new GKS-based package. New users should find some of the entry points convenient; for example, SPPS provides for automatic logarithmic scaling of plots, whereas the GKS standard does not. The FLUSH entry in the pre-GKS package has been given the name SFLUSH in SPPS, since FLUSH conflicted with a system entry of that name on many systems.
TAGUPW	Demonstration subroutine for the use of AGUPWRTX with AUTOGRPH.
TAREAS	Demonstration subroutine for the AREAS utility.
TAUTOG	Demonstration subroutine for the AUTOGRPH utility.
TCNQCK	Demonstration subroutine for the CONRCQCK utility.
TCNSMT	Demonstration subroutine for the CNRCSMTH utility.
TCNSUP	Demonstration subroutine for the CONRCSPR utility.
TCOLCV	Demonstration subroutine for the COLCONV utility.
TCONAN	Demonstration subroutine for the CONRAN utility.
TCONAQ	Demonstration subroutine for the CONRAQ utility.
TCONAS	Demonstration subroutine for the CONRAS utility.
TCONPA	Demonstration subroutine for the CONPACK utility.
TCONRE	Demonstration subroutine for the CONREC utility.
TDASHC	Demonstration subroutine for the DASHCHAR utility.
TDASHL	Demonstration subroutine for the DASHLINE utility.
TDASHP	Demonstration subroutine for the DASHSUPR utility.
TDASHS	Demonstration subroutine for the DASHSMTH utility.
TEZMAP	Demonstration subroutine for the EZMAP utility.
TEZMPA	Demonstration subroutine for the EZMAPA utility.
TGFLAS	Demonstration subroutine for the GFLASH utility.
TGRIDA	Demonstration subroutine for the GRIDAL utility.
THAFTO	Demonstration subroutine for the HAFTON utility.
THSTGR	Demonstration subroutine for the HISTGR utility.

TISOHR	Demonstration subroutine for the ISOSRFHR utility.
TISOSR	Demonstration subroutine for the ISOSRF utility.
TLBLBA	Demonstration subroutine for the LABELBAR utility.
TPLTCH	Demonstration subroutine for the PLOTCHAR utility.
TPWRTX	Demonstration subroutine for the PWRITX utility.
TPWRY	Demonstration subroutine for the PWRITY utility.
TPWRZI	Demonstration subroutine for the PWRZI utility.
TPWRZS	Demonstration subroutine for the PWRZS utility.
TPWRZT	Demonstration subroutine for the PWRZT utility.
TSOFTF	Demonstration subroutine for the SOFTFILL utility.
TSTITL	Demonstration subroutine for the STITLE utility.
TSRFAC	Demonstration subroutine for the SRFACE utility.
TSTRML	Demonstration subroutine for the STRMLN utility.
TTHREE	Demonstration subroutine for the THREED utility.
TVELVC	Demonstration subroutine for the VELVCT utility.

CGM Translator

CGMTRANS A program that reads either a pre-CGM NCAR metafile or an NCAR CGM and translates it into plot instructions for a given graphics device.

TRNSPPRT This file contains examples of the machine-dependent subroutines that need to be implemented on each machine for either the CGM translator or for the GRAPHC or FONTC programs. Fortran and/or C versions of these routines are provided as samples.

ARGGET: Get the requested argument from the command line.

BCLRED: Close a file opened by BOPRED for sequential binary reading.

BINCLS: Close a file opened for sequential binary writing. The file was opened by BINOPN.

BINOPN: Open a file for sequential binary writing.

BINRED: Transfer "COUNT" Fortran integers from an unformatted file that has been opened by BOPRED.

BINWRI: Transfer the contents of a buffer to the named file. Use a sequential unformatted write to perform the transfer. The file was opened by BINOPN.

BOFRED: Open a binary fontcap file for sequential unformatted reading. The file was created by the fontcap preprocessor using the local routine BINOPN.

BOPRED: Open a binary graphcap file for sequential unformatted reading. The file was created by the graphcap preprocessor using the local routine BINOPN.

CHRCLS: Close a file opened for sequential character reading. The file was opened by CHROPN.

CHROPN: Open a file for sequential unformatted reads. The file contains characters with a maximum of 80 characters per record.

CHRRED: Transfer from a character file to a Fortran character array. The file was opened by CHROPN.

FLUS: Dump device-dependent instructions.

FRPRMP: Send the "frame finished" prompt and wait for response (used only if the device is declared NON-BATCH).

IARGCT: Used only when optional command line processing is done, this routine returns as the function result the number of arguments on the command line.

INTHDL: Handles user interrupts.

MSSG: Puts out a message on the error unit that corresponds in format to the Status Message part of the "NCAR Computer Graphics Metafile Translator" section of this manual.

MTOPEN: Open a file for direct unformatted reading of 1440 byte records. The file is a metafile generated by BWI (see GKS 0A above).

READIT: Read a character string from standard input and put it into a FORTRAN 77 CHARACTER*1 array.

RECRED: Read in a metafile record using direct access I/O. The metafile is opened by the MTOPEN routine.

WRITIT: Write a character string to standard output.

Databases

EZMAPDAT	Continental outlines database for the EZMAP utility.
FONT1	Fontcap file for the default font used by the CGM interpreters.
FONT2	Fontcap file for the Hershey Cartographic Roman font.

FONT3	Fontcap file for the Hershey Cartographic Greek font.
FONT4	Fontcap file for the Hershey Simplex Roman font.
FONT5	Fontcap file for the Hershey Simplex Greek font.
FONT6	Fontcap file for the Hershey Simplex Script font.
FONT7	Fontcap file for the Hershey Complex Roman font.
FONT8	Fontcap file for the Hershey Complex Greek font.
FONT9	Fontcap file for the Hershey Complex Script font.
FONT10	Fontcap file for the Hershey Complex Italic font.
FONT11	Fontcap file for the Hershey Complex Cyrillic font.
FONT12	Fontcap file for the Hershey Duplex Roman font.
FONT13	Fontcap file for the Hershey Triplex Roman font.
FONT14	Fontcap file for the Hershey Triplex Italic font.
FONT15	Fontcap file for the Hershey Gothic German font.
FONT16	Fontcap file for the Hershey Gothic English font.
FONT17	Fontcap file for the Hershey Gothic Italian font.
FONT18	Fontcap file for the Hershey Math Symbols font.
FONT19	Fontcap file for the Hershey Symbol Set (Part 1) font.
FONT20	Fontcap file for the Hershey Symbol Set (Part 2) font.
GRAPHCAP	Master file of all graphcaps for the CGM translator.
PWRITXC1	Part I of card image representation of PWRITX Complex font digitization. Must be run through the PWRITXNT preprocessor to convert it to a binary form, which the PWRITX utility expects.
PWRITXC2	Part II of card image representation of PWRITX Complex font digitization. Must be run through the PWRITXNT preprocessor to convert it to a binary form, which the PWRITX utility expects.
PWRITXD1	Part I of card image representation of PWRITX Duplex font digitization. Must be run through the PWRITXNT preprocessor to convert it to a binary form, which the PWRITX utility expects.
PWRITXD2	Part II of card image representation of PWRITX Duplex font digitization. Must be run through the PWRITXNT preprocessor to convert it to a binary form, which the PWRITX utility expects.
RANFDAT	Data base for the demo plots produced form the BNCHMK file.

Database Initializers

FONTC	Preprocessor for the fontcap database file to convert them into a binary form needed by the CGM translator. Also required are several routines from the TRNSPPRT file.

GRAPHC Preprocessor for the graphcap database file to convert them into a binary form needed by the CGM translator. Also required are several routines from the TRNSPPRT file.

PWRITXNT Preprocessor for the PWRITX database files PWRITXC1, PWRITXC2, PWRITXD1, and PWRITXD2 to convert them into binary PWRITX database.

Examples

These files contain additional examples for the utilities listed. See the documentation for each utility for the output.

EAUTOG Extensive examples for the AUTOGRAPH utility. (These appear in the manual *AUTOGRAPH: A Graphing Utility.*)

ECONPA Extensive examples for the CONPACK utility.

EEZMAP Extensive examples for the EZMAP utility.

EEZMPA Additional example for the EZMAPA utility.

ELBLBA Additional example for the LABELBAR utility.

EMISC Miscellaneous examples: Examples for producing various color charts; a BIVAR example illustrating how to convert randomly-spaced data to gridded data for use with CONPACK, and an STITLE example.

EPLTCH Additional example for the PLOTCHAR utility.

ESOFTF Additional examples for the SOFTFILL utility.

Section IV: NCAR Computer Graphics Metafile Translator

Section IV: NCAR Computer Graphics Metafile Translator

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 NCAR Graphics produces 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 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 users have sets of plots that they want to store for later examination, the plots can be captured in a metafile. Also, if they want 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 of 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 — there is no support for graphical, input for example.

The Metafile Standard

The concept of metacode as 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 obtained 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.

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 standard 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, the user has to sequentially process all instructions in the metafile prior to the desired picture before plotting the desired picture. NCAR adopted 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 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. It is the case that the CGM elements capture much of the functionality of the GKS functions, but far from all — the picture segmentation of GKS is absent from the current set of CGM elements, for example. In addition, the CGM contains many elements that are not a part of the GKS standard — the CIRCULAR ARC elements, for example.

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 Section V of this manual. 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

The NCAR CGM Translator

NCAR Graphics creates, on output, a CGM file (conforming to the NCAR-specific record structure). To plot the encoded pictures in an NCAR CGM, SCD has written a metafile translator. This program takes as input a Computer Graphics Metafile and from it generates plot instructions specific to a particular computer graphics hardware device, such as a TEKTRONIX 4107 graphics terminal. The NCAR metafile translator is capable of accepting either the old NCAR metafiles (now officially known as pre-CGM NCAR metafiles), or NCAR CGM files. The GKS feature of pattern filling is not supported in this version of the NCAR CGM translator. Version 3.00 of the Fortran translator does support software simulation of hardware fill for devices that do not have hardware fill. The Polygon Control section under the "Graphcap Files" heading later in this section of the manual describes the keywords controlling polygon fill. The default limit for the maximum number of vertices accepted for software simulation of polygon fill is 12000. Software fill simulation can be time consuming. Facilities exist for disabling this feature — see the Polygon Control section under the "Graphcap Files" heading. All features (except pattern fill) that may be generated by the NCAR GKS 0A package are supported.

The NCAR CGM translator is table-driven in the sense that, to produce instructions for a specific output device, it reads in a device description table for that device and configures itself appropriately. It should be emphasized at this time that the device description tables apply only to devices that allow user access to the native command set — most entries in the table are commands specific to the output device. If the interface to a graphics device is a software interface, such as the subroutine-call interface to many workstations, then modifications to the translator code will be required. The C version of the NCAR CGM translator, distributed with the UNIX, VMS, and NCAR View packages, is designed to accommodate subroutine-call interfaces. As well as reading in a device description table, the translator reads in font tables describing the stroke sequences necessary to draw characters in the specified fonts. The device description tables are called graphcap files, and the font tables are called fontcap files. The graphcap and fontcap files read in by the translator are binary files that have been processed from original ASCII files. The original files are human-readable, whereas the binary files are only machine readable. To bring a new output device online, you need to fill in the blanks in the master ASCII graphcap form provided in the GRAPHCAP file on the distribution tape, or use a pre-existing device-specific graphcap from the GRAPHCAP file on the distribution tape if that is possible. This ASCII graphcap must be passed through the graphcap preprocessor (file GRAPHC on the distribution tape) in order to obtain the binary version of the graphcap as required by the translator. For details on how to create new graphcaps, see "Graphcap Files" later in this section. The situation with fontcap files is analogous to that for graphcaps. To create new font tables, see "Fontcap Files" later in this section. The ASCII fontcap files have to be processed by the

fontcap preprocessor (file FONTC on the distribution tape) to obtain the binary version required by the translator. There are 41 supported graphcaps on the Version 3.00 distribution tape. The graphcaps are summarized in the "Supported Graphcaps in Version 3.00" table under the "Graphcap Files" heading later in this section of the manual.

Installation

The NCAR CGM translator consists of three major programs and two support files. They are as follows:

Programs:	fontcap preprocessor
	graphcap preprocessor
	CGM translator
Data Files:	graphcap file
	fontcap file

The CGM translator is the program which will, after correct installation, read CGM metafiles and create specific graphic device commands. The files on the distribution tape that pertain to the implementation of the translator are:

CGMTRANS	Fortran source for the Fortran NCAR CGM translator.
GRAPHC	Fortran source for the graphcap preprocessor.
FONTC	Fortran source for the fontcap preprocessor.
GRAPHCAP	Collection of ASCII graphcap files.
FONTCAP	ASCII fontcap file.
LOCAL	Examples of local implementations of machine-specific routines to be written by the implementor.
TRNSPPRT	Examples of local implementations of machine-specific routines to be written by the implementor.

Installation Steps

1. Write the installation-dependent routines.

   The installation-dependent routines are described in Appendix A of Section II of this manual and under the "Required Locally Implemented Subroutines" heading in this section. These routines deal with I/O and bit manipulation. Some of the routines are the same as those required by the NCAR GKS 0A implementation. Examples of some machine implementations of these routines are found in files LOCAL and TRNSPPRT on the distribution tape. It should be emphasized that the examples in LOCAL and TRNSPPRT are not portable — care must be taken to guarantee that the implementations of these routines satisfy the functional descriptions in Appendix A of Section II of this manual, and under the "Required Locally Implemented Subroutines" heading in this manual section.

2. Create and execute the fontcap preprocessor.

   This module is created from the local files in Step 1 above and from FONTC. Execute this module. You will be asked to enter names for the ASCII input fontcap file and for the binary output file. After each such name pair you will be asked if you want to process another fontcap. You can process all fontcaps by entering an input script with the

appropriate responses to the preprocessor — the preprocessor reads from standard input and writes to standard output. To create your own fontcap files, see "Fontcap Files" later in this section. The binary fontcap produced will be read in by the translator at translator execution time.

3. Create and execute the graphcap preprocessor.

   This module is created from the local files in Step 1 above, and from GRAPHC on the distribution tape. Execute the module. You will be asked to enter names for the ASCII input graphcap file and for the binary output file. After each such name pair you will be asked if you want to process another graphcap. You can process all graphcaps by entering an input script with the appropriate responses to the preprocessor — the preprocessor reads from standard input and writes to standard output. The graphcap files are on the distribution tape in file GRAPHCAP. The graphcaps are all concatenated into one file. To use a graphcap file from GRAPHCAP as input for the graphcap preprocessor, it will have to be stripped out of GRAPHCAP; each individual graphcap in GRAPHCAP begins with a line of the form

   NCAR_FILE_START name.gc

   and ends with a line of the form

   NCAR_FILE_END name.gc

   The *name* in the above lines is a name indicating the device that the enclosed graphcap describes. The two special separator lines for each graphcap must be stripped off before using the graphcap as input to the graphcap preprocessor in GRAPHC. It is an easy task to write a procedure that splits the master graphcap file into the desired individual graphcaps.

   If a graphcap does not exist in GRAPHCAP for a device you want to bring online, you will have to fill in the master graphcap form for that device as per the instructions in "Graphcap Files" in this section of the manual. A binary version for a given graphcap need be created only once, unless changes are made to it.

4. Specify how the graphcap filenames will be supplied to the translator.

   Supplying the graphcap filename to the translator will depend on the specific host environment. The GRAPHCAP file is read in by the translator in subroutine REDDEV, and the locally implemented routine BOPRED is used to do the OPEN. In the default state (CGMTRANS as supplied on the distribution tape), the graphcap filename is stored in the CHARACTER*1 array GRCAPN, and this is initialized in BLOCKDATA TRNDAT by the statement

```
DATA (GRCAPN(II),II=1,9)/'G','R','A','P','H','C','A','P',' '/
```

   Several possibilities exist for attaching the input graphcap file:

   - Place the desired graphcap on the file system with filename GRAPHCAP. This is not a desirable solution, since you typically want to use the translator to drive several devices, in which case it is advisable to give graphcaps mnemonic names, such as t4107 for a TEKTRONIX 4107 driver, for example.

- Modify subroutine REDDEV so that it requests the graphcap filename from the user. Note that if BOPRED is used to open the file, the filename must be stored in a CHARACTER*1 array and the name must be terminated with a blank character.
- Hardwire the graphcap filename by redefining the DATA statement in BLOCK-DATA TRNDAT. This has the same disadvantages as the first option.
- Modify the CGMTRANS subroutine GARG so that the graphcap filename can be passed on the command line (this is of course only possible if a command line parser is available).
- On most UNIX systems, the environment variable graphcap can be set to the path-name of the input graphcap file. BINOPN can then be written so that it checks whether the input filename graphcap is an environment variable, and if it is, BINOPN will open the appropriate file. The example BINOPN in file TRNSPPRT uses this method.
- On batch machines, pre-assign a Fortran logical unit to the input graphcap file, and call BINOPN in REDDEV with the appropriate unit number.

5. Specify how the fontcap filename will be supplied to the translator.

   A fontcap file is read by the translator in subroutine REDFNT. REDFNT calls the locally implemented subroutine BOFRED to open its fontcap file. The user should implement BOFRED so that font numbers 1 through 20 are used to open the fontcaps FONT1 through FONT20 on the distribution tape. It is not necessary to supply more than the one font in file FONT1.

6. Specify how the input metafile name will be supplied to the translator.

   Subroutine FNDARG is used to attach the metafile for reading. The LOGICAL variable METIPT is used to control whether the metafile name will be requested by a Fortran read from the standard input, or whether the metafile name will be read from the command line. METIPT is initialized to .FALSE. in BLOCKDATA TRNDAT; setting METIPT to .FALSE. provides that the metafile name will be read from the command line. FNDARG invokes GARG to do the Fortran or command line read. If a Fortran read is implemented (by setting METIPT to .TRUE.), the locally-implemented routines IARGCT and ARGGET can be dummy routines. Otherwise, IARGCT and ARGGET are used to read the arguments from the command line. In its default state, the first command line argument to the translator will be used for the metafile name. If an optional second argument is supplied, it will be used as a record pointer into the input metafile to indicate the record where translation will begin, and only one frame will be translated in this case. The use of the optional second argument is helpful when the translator is invoked from an interactive metafile editor.

7. When binary graphcap and fontcap files have been created, and you have an input metafile available, compile and execute the translator. The translator uses the locally-implemented routine FLUS to send its output to the output device. The unit number in calls to FLUS is defaulted to 1 (the standard output on most UNIX systems).

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 to produce a binary version of the graphcap, which is in turn used by an NCAR CGM translator at execution time. The details of how graphcaps fit into the overall picture of metafile translation are discussed at the beginning of this section of this manual.

This discussion that follows 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, modifying an existing graphcap to produce a different color table, change the frame finished prompt, modify the picture orientation or select a different line type are tasks easily performed. 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 you cannot utilize the graphcap concept. The ambitious installer may want to undertake making the necessary changes to the translator code itself in order to implement the Fortran 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 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. A complete description of all graphcap keywords appears 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 can use the master form, which is part of file GRAPHCAP on the distribution tape. This master file contains keywords for all currently supported graphcap functions.

The keyword definitions are of four types:

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, and 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. The

legal ASCII characters are those which appear in the "CHAR" column of the table in Appendix A at the end of this section. Each character in the keyword string definition is separated by one or more blanks or a new line. The 3-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 8 bits of a Fortran type INTEGER. There are 7 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 it is desired 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 XY-coordinate pair is to be inserted into the keyword string definition.

Understanding the use of these special characters will be aided by examining their use in some of the graphcaps supplied in file GRAPHCAP 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; however, they may cross line boundaries. For the purpose of 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 bundles, 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.

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 this 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.

Supported Graphcaps in Version 3.00

The following table lists all the graphcaps supported in the Version 3.00 release of NCAR Graphics.

Supported Graphcaps in Version 3.00

adm5	ADM5 Graphics terminal with DEC RG1000 graphics board
aed.a	AED512 in ASCII mode
aed.b	AED512 in binary mode
balsml.g	HI DMP-29 in small chart mode
form	Generic graphcap template containing all defined keywords
hp2648a	HP2648a in binary mode
hp150	HP150 personal computer
hp7475a	HP7475a six pen plotter
hp7510a	HP7510a color film recorder
hplj100l	HP LaserJet, LaserJet Plus, LaserJet Series II, LaserJet 500 Plus, and so on, in landscape mode at 100 dots per inch
hplj100p	HP LaserJet, LaserJet Plus, LaserJet Series II, LaserJet 500 Plus, and so on, in portrait mode at 100 dots per inch
hplj150l	HP LaserJet, LaserJet Plus, LaserJet Series II, LaserJet 500 Plus, and so on, in landscape mode at 150 dots per inch
hplj150p	HP LaserJet, LaserJet Plus, LaserJet Series II, LaserJet 500 Plus, and so on, in portrait mode at 150 dots per inch
hplj300l	HP LaserJet, LaserJet Plus, LaserJet Series II, LaserJet 500 Plus, and so on, in landscape mode at 300 dots per inch
hplj300p	HP LaserJet, LaserJet Plus, LaserJet Series II, LaserJet 500 Plus, and so on, in portrait mode at 300 dots per inch
hplj75l	HP LaserJet, LaserJet Plus, LaserJet Series II, LaserJet 500 Plus, and so on, in landscape mode at 75 dots per inch

(continued on next page)

Supported Graphcaps in Version 3.00 (continued)

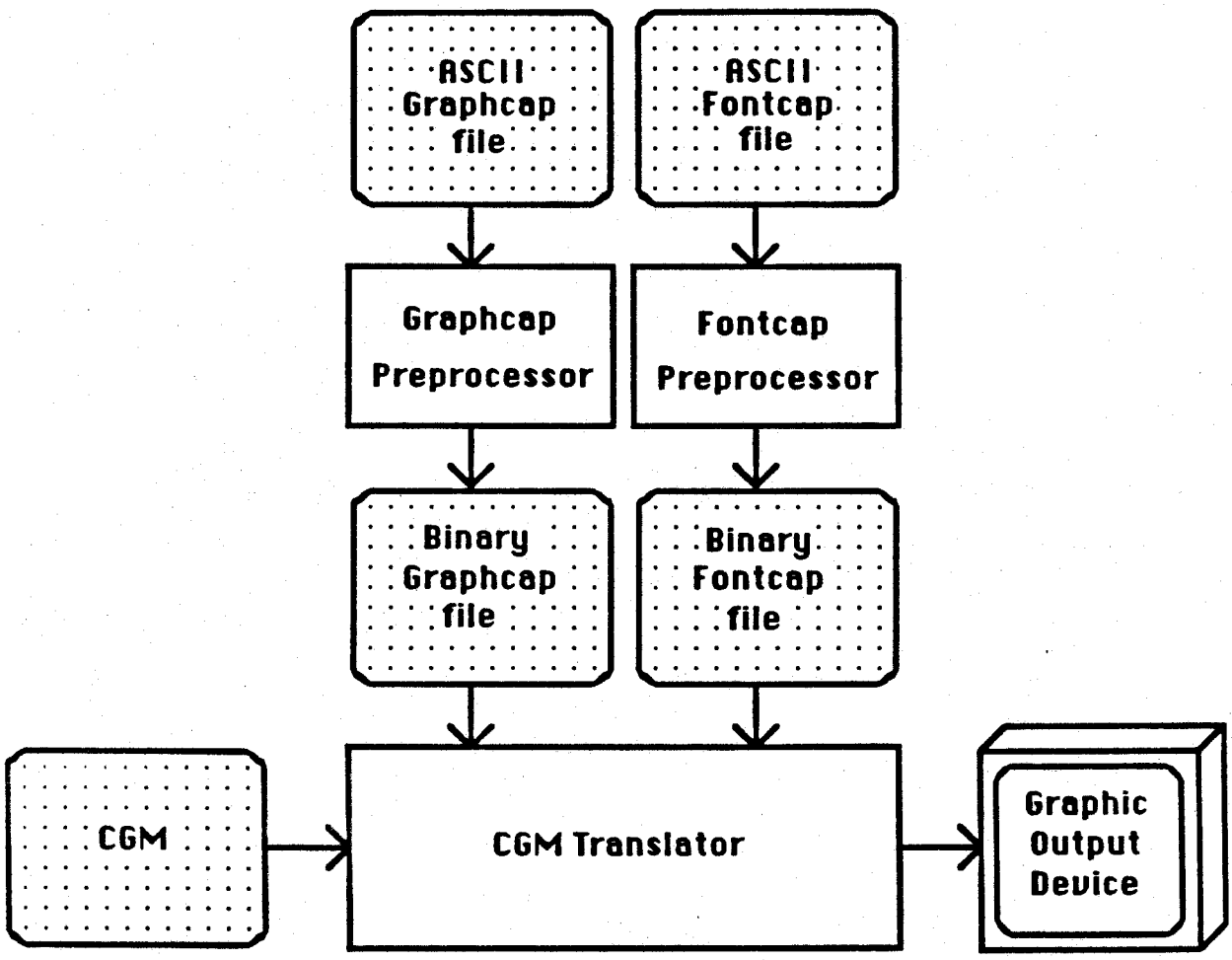
hplj75p	HP LaserJet, LaserJet Plus, LaserJet Series II, LaserJet 500 Plus, and so on, in portrait mode at 75 dots per inch
hpljo75l	The original LaserJet family at 75 dots per inch in landscape mode
hpljo75p	The original LaserJet family at 75 dots per inch in portrait mode
imagen	IMAGEN 8/300 laser printer in graphics landscape mode
imagen.port	IMAGEN 8/300 laser printer in graphics portrait mode
pc.mono	PCPLOT on IBM and compatible PC's
ps.mono	spooled PostScript devices
qms800	QMS800 Laser printer
r6211	RAMTEK 6211 in TEKTRONIX compatible mode
s100	SELINAR HiREZ100 graphics terminal
t4006	TEKTRONIX4006
t4010	TEKTRONIX 4010 and TEKTRONIX 4012
t4025	TEKTRONIX 4025
t4105	TEKTRONIX 4105
t4107	TEKTRONIX 4107 (same as TEKTRONIX 4207); cell arrays are simulated and instructions are put into segment 1
t4107.seg	TEKTRONIX 4107 (same as TEKTRONIX 4207); scan lines used for cell arrays, instructions not put in segment
t4115	TEKTRONIX 4115 cell arrays are simulated and instructions are put into segment 1
t4115.seg	TEKTRONIX 4115 scan lines used for cell arrays, instructions not put in segment
tal1590	Talaris 1590 in TEKTRONIX mode.
tekalike	Tekalike on the Apple Macintosh
versaterm	VersaTerm on the Apple Macintosh
vt100	VT100 with DEC VT640 retrofit
vt125	VT125
vt220	VT220 with SELINAR SG220 graphcis retrofit board
vt330	VT330

There are 20 supported fontcaps on the Version 3.00 distribution tape. These fontcaps appear in files FONTn, n=1,...,20, on the distribution tape. For a complete display of all of these consult the "Fontcap Files" section of this chapter.

A schematic of how the translator works is shown on the next page. Once a binary version of a given graphcap or fontcap is created, it need not be regenerated unless changes are made to the ASCII original. It is assumed that the graphcap and fontcap preprocessors will be run on the same host that the translator is run on.

Several additions and changes have been made to the graphcap file definitions and to the graphcap preprocessor (file GRAPHC on the distribution tape) for Version 3.00. Version 3.00 binary graphcaps are not compatible with Version 2.00 software. Installers of Version 3.00 will have to regenerate all binary graphcaps. The graphcap preprocessor has

How the CGM Translator Works



been modified to generate an arbitrary number of binary graphcaps with a single execution.

The graphcap concept is designed to make it easy to bring online new devices that have user access to the device commands (such as most graphics terminals). The translator is coded to produce device instructions. If the interface to a graphics device, such as many graphics workstations, is via subroutine calls, then most likely the actual code in the translator will have to be modified for installation, or the C language implementation of the NCAR CGM translator should be used.

Formatting and Encoding Coordinates

The process of creating coordinates, color or intensities, and other parameters that are part of a device instruction is normally not simple. Many devices require bits from various parts of the data values be packed together to form the device instruction parameters. For example, the TEKTRONIX 40XX series requires that the first word of a device coordinate parameter contain 5 bits from the upper part of the Y coordinate packed with some flagging information. Other devices require parts of the X and Y coordinate be packed together. Describing this formatting information to the translator is provided for 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. The TEKTRONIX 41XX terminal series requires that the bit stream from the formatting process be converted into a set of printable characters for example.

Many devices require that parameters be converted in these two steps. The present discussion will explain 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 is 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 4 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 (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, for example, 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 has been processed. Suppose you had a table defining XY coordinates. The X's and Y's 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 3 entries for each color definition, so define how those 3 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 will 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 X and Y 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 0's and 1's 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 an 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 an 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 interpreted as a whole number, depends on the corresponding FLOATING_INFO keyword. The FLOATING_INFO type keywords provide 4 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 <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 hand 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 <return>) is required to continue. This string is used only if DEVICE_BATCH is set FALSE. Typically, the prompt is displayed in Graphics mode. On small single-page plotters you may wish 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.

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 discussion.

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 discussion.

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 under "Formatting and Encoding Coordinates" earlier in this section. Two examples follow. They describe the required bit positions as P for parity, X*n* for an X bit and Y*n* 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 follows:

```
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 will 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 4 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" discussion earlier in this section.

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 4 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 "Formatting and Encoding Coordinates" discussion earlier in this section.

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 4 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 discussion on "Formatting and Encoding Coordinates" earlier in this section.

Example

This 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 4 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 of the CGM Standard document).

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 discussion on "Formatting and Encoding Coordinates" earlier in this section.

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 discussion on "Formatting and Encoding Coordinates" earlier in this section.

Example

This example uses a 4-bit color index (0 to 15 decimal). The P represents parity bit; 0's (zero) and 1's 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 4 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 though 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 discussion on "Formatting and Encoding Coordinates" earlier in this section.

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 or 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 discussion on "Formatting and Encoding Coordinates" earlier in this section.

Example

Assume a RGB color model with 100 map entries. The formatter will have 300 words passed to it (3 per entry). The device uses 8 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 will also have 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 4 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 3-tuples.
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 which 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 blown up 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 polylines, 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 is used to initiate 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 found 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 which 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 linewidth. The translator first checks to see if there is a linewidth command available on the device (the graphcap entry LINE_WIDTH_INSTRUCTION_START is non-empty). If there is no line width instruction on the device, CGM 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 CGM LINE WIDTH elements are interpreted as linewidth scale factors. If the device does have a line width instruction, then a linewidth 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 the above equation "new_scale_value" is the floating 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 WITH 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 discussion on "Formatting and Encoding Coordinates" earlier in this section.

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 4 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.

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 4 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 value multiplied into the computed device line width units before it is sent to 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 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, and 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

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

In addition, the graphcap preprocessor 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.

Graphical Text Control

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 what bundle tables are, 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 installer 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 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

This section controls the generation of polygons, closed objects that 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: If a device has a hardware fill instruction (the POLYGON_INSTRUCTION_START keyword has been defined), then hardware fill will be done for all polygons having less than one plus the number of vertices specified by the POLYGON_MAXIMUM_POINTS keyword. In all other cases, software fill will automatically be invoked unless the POLYGON_SIMULATE keyword is set to FALSE. If the POLYGON_SIMULATE keyword is set to FALSE, then only the boundaries of the polygons will be drawn (in the current fill color) whenever software fill would have otherwise been invoked. This keyword is designed to disable automatic software fill on devices that have no hardware fill instruction.

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 virtual device coordinate space 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))

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 that 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 virtual device coordinate range 0 to 32767.

Type: Decimal

Default: 300

Maximum number of entries: 1

Raster Control

This section controls 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 what cell arrays are, 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. In the case that 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. 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.

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 (in pixel space) 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 (in pixel space) 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 (in pixel space) 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 (in pixel space) 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 upon the RASTER_HORIZONTAL_INSTRUCTION_START definition, the input array may contain a color index for every pixel, or it may be runlength encoded (an array containing pairs of values, the first element of the pair specifying a runlength which is a count specifying a number of pixels, and the second element specifying a color index). Consult the "Formatting and Encoding Coordinates" discussion earlier in this section.

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 runlength encoded pixel values. The first value is the pixel count, and the second is the color index. Each set of runlength 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 4 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 "Formatting and Encoding Coordinates" discussion earlier in this section.

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 4 elements per row.

Default: Null

Maximum number of entries: 40 (10 rows with 4 elements per row)

RASTER_VECTOR_COUNT_ENCODING

Description: The encoding scheme used for the device raster data. See the "Formatting and Encoding Coordinates" discussion earlier in this section.

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. The details of how fontcaps fit into the overall picture of metafile translation are discussed in the beginning of this section of this manual.

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. A complete description of all fontcap keywords appears 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 a maximum count specified by the associated keyword. Keyword definitions are allowed a maximum of 80 columns per line; however, they may cross line boundaries. A new line is equivalent to a space for separating definition entities.

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.

Note: 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 the 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 in font units of each character in the font. For proportionally spaced fonts, this is the width of the widest character, including white space.

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 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 installer 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 vertical center 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. Extenders drop down to this level.

Type: Decimal

Maximum number of entries: 1

FONT_TYPE

Description: Flags the font type. 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 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 indicator.

Type: Decimal

Maximum number of entries: 1

PAINT_END_LEN

Description: The number of bits in the 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 preceded by a CHAR keyword. The definition of a character will differ depending upon 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 that 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. 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. 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 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 6 bits. In the second example, the X and Y lengths are 8 bits. The PEN and PAINT are 1 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 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:

NCAR GKS font number	Table column heading	Font name
1	1	DEFAULT
-2	M2	HERSHEY:CARTOGRAPHIC_ROMAN
-3	M3	HERSHEY:CARTOGRAPHIC_GREEK
-4	M4	HERSHEY:SIMPLEX_ROMAN
-5	M5	HERSHEY:SIMPLEX_GREEK
-6	M6	HERSHEY:SIMPLEX_SCRIPT
-7	M7	HERSHEY:COMPLEX_ROMAN
-8	M8	HERSHEY:COMPLEX_GREEK
-9	M9	HERSHEY:COMPLEX_SCRIPT
-10	M10	HERSHEY:COMPLEX_ITALIC
-11	M11	HERSHEY:COMPLEX_CYRILLIC
-12	M12	HERSHEY:DUPLEX_ROMAN
-13	M13	HERSHEY:TRIPLEX_ROMAN
-14	M14	HERSHEY:TRIPLEX_ITALIC
-15	M15	HERSHEY:GOTHIC_GERMAN
-16	M16	HERSHEY:GOTHIC_ENGLISH
-17	M17	HERSHEY:GOTHIC_ITALIAN
-18	M18	HERSHEY:MATH_SYMBOLS
-19	M19	HERSHEY:SYMBOL_SET1
-20	M20	HERSHEY:SYMBOL_SET2

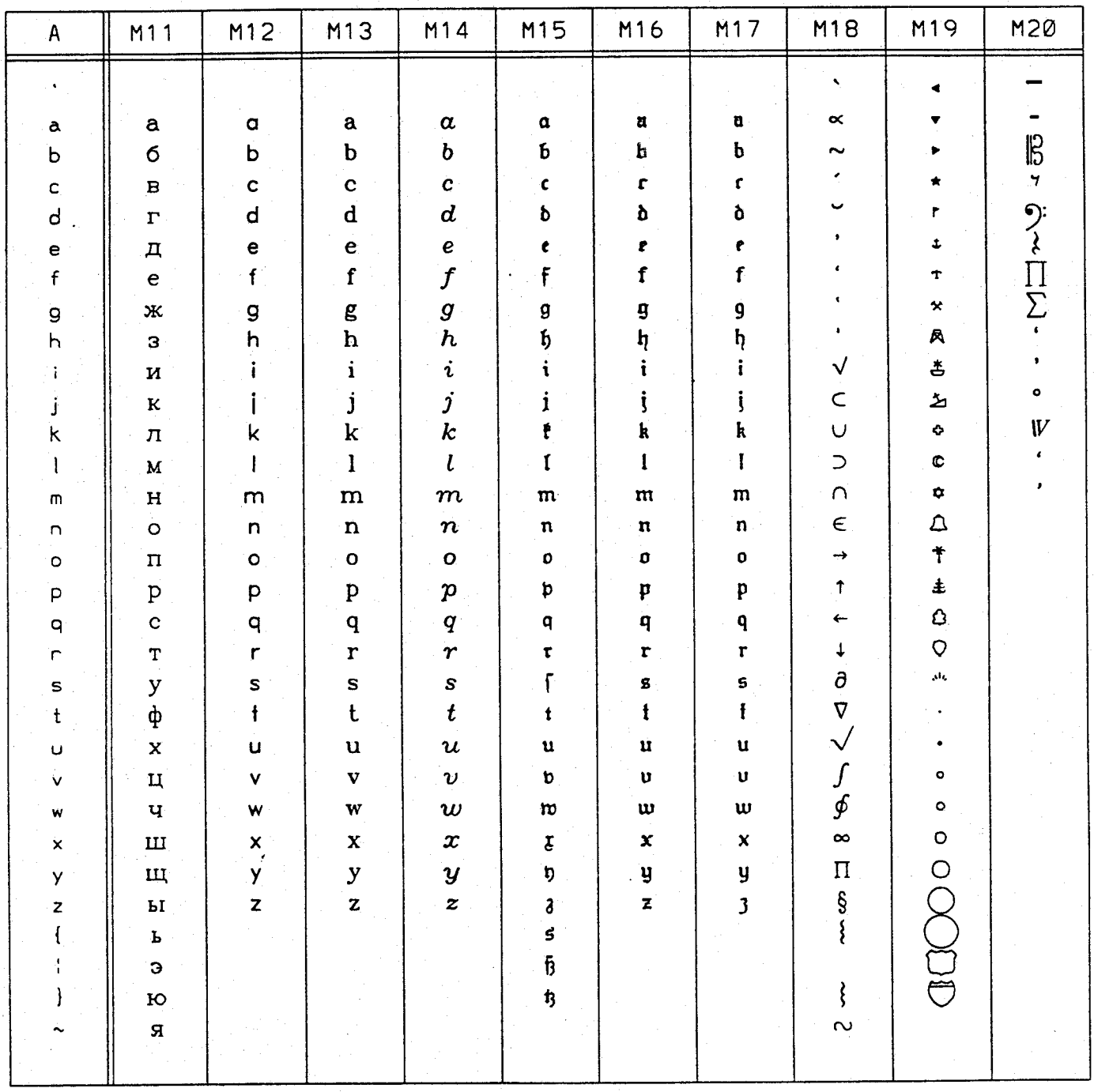
A	1	M2	M3	M4	M5	M6	M7	M8	M9	M10
spc										
!	!	!	!	!	!	!	!	!	!	!
"	"	"	"	"	"	"	"	"	"	"
#	#	#	#	#	#	#	#	#		
$	$	$	$	$	$	$	$	$	$	$
%	%			%	%		%	%		
&	&	&	&	&	&	&	&	&	&	&
'	'	'	'	'	'	'	'	'	'	'
(	(	(	(	(	(	(	(	(	(	(
)	)	)	)	)	)	)	)	)	)	)
*	*	*	*	*	*	*	*	*	*	*
+	+	+	+	+	+	+	+	+	+	+
,	,	,	,	,	,	,	,	,	,	,
-	-	-	-	-	-	-	-	-	-	-
.	.	.	.	.	.	.	.	.	.	.
/	/	/	/	/	/	/	/	/	/	/
0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9
:	:	:	:	:	:	:	:	:	:	:
;	;	;	;	;	;	;	;	;	;	;
<	<	<	<	<	<	<	<	<		
=	=	=	=	=	=	=	=	=	=	=
>	>			>	>	>	>	>		
?	?	?	?	?	?	?	?	?	?	?

A	1	M2	M3	M4	M5	M6	M7	M8	M9	M10
@	@						@			@
A	A	A	A	A	A	$\mathcal{A}$	A	A	$\mathcal{A}$	*A*
B	B	B	B	B	B	$\mathcal{B}$	B	B	$\mathcal{B}$	*B*
C	C	C	X	C	X	$\mathcal{C}$	C	X	$\mathcal{C}$	*C*
D	D	D	Δ	D	Δ	$\mathcal{D}$	D	Δ	$\mathcal{D}$	*D*
E	E	E	E	E	E	$\mathcal{E}$	E	E	$\mathcal{E}$	*E*
F	F	F	Φ	F	Φ	$\mathcal{F}$	F	Φ	$\mathcal{F}$	*F*
G	G	G	Γ	G	Γ	$\mathcal{G}$	G	Γ	$\mathcal{G}$	*G*
H	H	H	H	H	H	$\mathcal{H}$	H	H	$\mathcal{H}$	*H*
I	I	I	I	I	I	$\mathcal{I}$	I	I	$\mathcal{I}$	*I*
J	J	J		J	ϑ	$\mathcal{J}$	J	ϑ	$\mathcal{J}$	*J*
K	K	K	K	K	K	$\mathcal{K}$	K	K	$\mathcal{K}$	*K*
L	L	L	Λ	L	Λ	$\mathcal{L}$	L	Λ	$\mathcal{L}$	*L*
M	M	M	M	M	M	$\mathcal{M}$	M	M	$\mathcal{M}$	*M*
N	N	N	N	N	N	$\mathcal{N}$	N	N	$\mathcal{N}$	*N*
O	O	O	O	O	O	$\mathcal{O}$	O	O	$\mathcal{O}$	*O*
P	P	P	Π	P	Π	$\mathcal{P}$	P	Π	$\mathcal{P}$	*P*
Q	Q	Q	Θ	Q	Θ	$\mathcal{Q}$	Q	Θ	$\mathcal{Q}$	*Q*
R	R	R	P	R	P	$\mathcal{R}$	R	P	$\mathcal{R}$	*R*
S	S	S	Σ	S	Σ	$\mathcal{S}$	S	Σ	$\mathcal{S}$	*S*
T	T	T	T	T	T	$\mathcal{T}$	T	T	$\mathcal{T}$	*T*
U	U	U	ϒ	U	ϒ	$\mathcal{U}$	U	ϒ	$\mathcal{U}$	*U*
V	V	V		V	ς	$\mathcal{V}$	V	ς	$\mathcal{V}$	*V*
W	W	W	Ω	W	Ω	$\mathcal{W}$	W	Ω	$\mathcal{W}$	*W*
X	X	X	Ξ	X	Ξ	$\mathcal{X}$	X	Ξ	$\mathcal{X}$	*X*
Y	Y	Y	Ψ	Y	Ψ	$\mathcal{Y}$	Y	Ψ	$\mathcal{Y}$	*Y*
Z	Z	Z	Z	Z	Z	$\mathcal{Z}$	Z	Z	$\mathcal{I}$	*Z*
[	[						[	[		
\	\			\	\	\	\	\		
]	]						]	]		
^	^			^	^	^	^	^		
_	_									

A	1	M2	M3	M4	M5	M6	M7	M8	M9	M10
`	`			`	`	`	`	`		
a	a			a	α	a	a	α	a	α
b	b			b	β	b	b	β	b	b
c	c			c	χ	c	c	χ	c	c
d	d			d	δ	d	d	δ	d	d
e	e			e	ε	e	e	ε	e	e
f	f			f	ϕ	f	f	ϕ	f	f
g	g			g	γ	g	g	γ	g	g
h	h			h	η	h	h	η	h	h
i	i			i	ι	i	i	ι	i	i
j	j			j	φ	j	j	φ	j	j
k	k			k	κ	k	k	κ	k	k
l	l			l	λ	l	l	λ	l	l
m	m			m	μ	m	m	μ	m	m
n	n			n	ν	n	n	ν	n	n
o	o			o	o	o	o	o	o	o
p	p			p	π	p	p	π	p	p
q	q			q	θ	q	q	θ	q	q
r	r			r	ρ	r	r	ρ	r	r
s	s			s	σ	s	s	σ	s	s
t	t			t	τ	t	t	τ	t	t
u	u			u	υ	u	u	υ	u	u
v	v			v	ϵ	v	v	ϵ	v	υ
w	w			w	ω	w	w	ω	w	w
x	x			x	ξ	x	x	ξ	x	x
y	y			y	ψ	y	y	ψ	y	y
z	z			z	ζ	z	z	ζ	z	z
{	{						{	{		
|	|	|	|	|	|	|	|	|		
}	}						}	}		
~	~									

A	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20
spc										
!	!	!	!	!	!	!	!	!	▫	fi
"	"	"	"	"	"	"	"	"	▽	fl
#	#							#	□	ffi
$	$	$	$	$	$	$	$	Σ	♠	ffl
%	%							%	♡	ı
&	&	&	&	&	&	&	&	&	◇	ȷ
'	'	'	'	'	'	'	'	'	♣	ff
(	(	(	(	(	(	(	(	(	♧	fi
)	)	)	)	)	)	)	)	)	✱	fl
*	*	*	*	*	*	*	*	*	▲	ffi
+	+	+	+	+	+	+	+	+	●	ffl
,	,	,	,	,	,	,	,	,	▮	ı
-	–	–	–	–	–	–	–	–	^	ȷ
.	.	.	.	.	.	.	.	.	ˆ	♀
/	/	/	/	/	/	/	/	/	ˇ	⊕
0	0	0	0	0	0	0	0	0	˘	♂
1	1	1	1	1	1	1	1	1	˙	♃
2	2	2	2	2	2	2	2	2	˚	♄
3	3	3	3	3	3	3	3	3	¸	♁
4	4	4	4	4	4	4	4	4	˛	Ψ
5	5	5	5	5	5	5	5	5	S	Ƥ
6	6	6	6	6	6	6	6	6	~	☾
7	7	7	7	7	7	7	7	7	˝	☌
8	8	8	8	8	8	8	8	8	˛	*
9	9	9	9	9	9	9	9	9	—	Ω
:		:	:	:	:	:	:	:	—	♉
;		;	;	;	;	;	;	;	/	♈
<								<	/	♉
=	Й	=	=	=	=	=	=	=	|	♯
>	й							>	\	♋
?	Ъ	?	?	?	?	?	?	?	/	Ω

A	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20
@	Ъ							@	–	
A	А	A	A	*A*	𝔄	𝕬	𝔸	⊥	ˋ	Ω
B	Б	B	B	*B*	𝔅	𝕭	𝔹	∠	–	₥
C	В	C	C	*C*	ℭ	𝕮	ℂ	∴	ˊ	↗
D	Г	D	D	*D*	𝔇	𝕯	𝔻	·	ˆ	↘
E	Д	E	E	*E*	𝔈	𝕰	𝔼	—	ˇ	=
F	Е	F	F	*F*	𝔉	𝕱	𝔽	/	˘	✳
G	Ж	G	G	*G*	𝔊	𝕲	𝔾	*	˙	·
H	З	H	H	*H*	ℌ	𝕳	ℍ	•	˚	ˋ
I	И	I	I	*I*	ℑ	𝕴	𝕀	°	¸	ˋ
J	К	J	J	*J*	𝔍	𝕵	𝕁	╱	˛	0
K	Л	K	K	*K*	𝔎	𝕶	𝕂	≦	˜	0
L	М	L	L	*L*	𝔏	𝕷	𝕃	≧	˝	•
M	Н	M	M	*M*	𝔐	𝕸	𝕄	†	¯	#
N	О	N	N	*N*	𝔑	𝕹	ℕ	‡	˘	♮
O	П	O	O	*O*	𝔒	𝕺	𝕆	〈	¸	♭
P	Р	P	P	*P*	𝔓	𝕻	ℙ	〉	˛	♯
Q	С	Q	Q	*Q*	𝔔	𝕼	ℚ	—	˝	-
R	Т	R	R	*R*	ℜ	𝕽	ℝ	═	ˆ	×
S	У	S	S	*S*	𝔖	𝕾	𝕊	±	ˇ	𝄞
T	Ф	T	T	*T*	𝔗	𝕿	𝕋	∓	˜	𝄢
U	Х	U	U	*U*	𝔘	𝖀	𝕌	×	¨	𝄡
V	Ц	V	V	*V*	𝔙	𝖁	𝕍	·	˙	·
W	Ч	W	W	*W*	𝔚	𝖂	𝕎	÷	˚	ˋ
X	Ш	X	X	*X*	𝔛	𝖃	𝕏	≠	˚	ˋ
Y	Щ	Y	Y	*Y*	𝔜	𝖄	𝕐	≡	˚	ˋ
Z	Ы	Z	Z	*Z*	ℨ	𝖅	ℤ	⊙	˚	ˋ
[	Ь							[	˚	ˋ
\	Э							/	˚	ˋ
]	Ю							]	˚	ˋ
^	Я							^	˚	ˋ
_								∃	˚	ˋ



A	M11	M12	M13	M14	M15	M16	M17	M18	M19	M20
`								`	◂	—
a	а	a	a	*a*	ɑ	ɑ	ɑ	∝	▾	-
b	б	b	b	*b*	ɓ	ɓ	b	~	▸	≡
c	в	c	c	*c*	ɕ	ɾ	ɾ	ˋ	⋆	ɾ
d	г	d	d	*d*	ɖ	ð	ð	ˇ	ɾ	ʔ
e	д	e	e	*e*	ɛ	ɐ	ɐ	˙	⊥	≈
f	е	f	f	*f*	ɟ	f	f	ˊ	⊤	∏
g	ж	g	g	*g*	g	g	g	¨	×	∑
h	з	h	h	*h*	ɦ	ɧ	ɧ	˘	⋔	˙
i	и	i	i	*i*	i	i	i	√	⊥	˚
j	к	j	j	*j*	ɟ	ɟ	ɟ	⊂	⊾	˛
k	л	k	k	*k*	ƙ	k	k	∪	⌖	W
l	м	l	l	*l*	ɭ	l	l	⊃	⌀	˜
m	н	m	m	*m*	m	m	m	∩	✡	˝
n	о	n	n	*n*	n	n	n	∈	⏃	
o	п	o	o	*o*	o	ɑ	o	→	☨	
p	р	p	p	*p*	ᵽ	ᵽ	p	↑	⏄	
q	с	q	q	*q*	q	q	q	←	⏢	
r	т	r	r	*r*	ɾ	r	r	↓	⍜	
s	у	s	s	*s*	ʃ	ʂ	s	∂	⍝	
t	ф	t	t	*t*	t	t	t	∇	·	
u	х	u	u	*u*	u	u	u	√	•	
v	ц	v	v	*v*	ⱱ	ʊ	ʊ	∫	∘	
w	ч	w	w	*w*	ɯ	ɯ	ɯ	∮	○	
x	ш	x	x	*x*	ɤ	x	x	∞	○	
y	щ	y	y	*y*	ɳ	ɣ	ɣ	Π	○	
z	ы	z	z	*z*	ȝ	z	ȝ	§	◯	
{	ь				ʂ			≀	◯	
|	э				ß				◯	
}	ю				ʒ			≀	⬭	
~	я							∾	⬭	

Required Locally Implemented Subroutines

The graphcap preprocessor, the fontcap preprocessor, and the CGM translator all invoke low-level support routines from a collection of such subroutines that are machine-specific and need to be supplied locally by the installer.

Three of the routines, I1MACH, GBYTES, and SBYTES, can be found in "NCAR Graphics Package Installation," Section II of this manual; the descriptions for all other required support routines will be given in this section.

Examples of I1MACH, GBYTES, and SBYTES are contained in file LOCAL on the distribution tape; examples of the other required support routines are in file TRNSPPRT on the distribution tape. It must be emphasized that these examples are machine-specific and are meant as examples only. The installer must take care to make certain that the implementations of the support routines satisfy the functional descriptions given below.

The support routines described below are separated into those required by the translator, and those required by the fontcap and graphcap preprocessors.

Routines Required by the Translator

This part describes the routines required by the Fortran NCAR CGM translator (file CGMTRANS) at each installation. All variable types are given in terms of FORTRAN 77. In the descriptions below, "allok" represents no errors occurring during the execution of the described routine.

The routines I1MACH, GBYTES, and SBYTES are required but not discussed here. Their description will be found in Appendix A of Section II of this manual.

ARGGET (NUMBER, STRING)

ARGGET gets the requested argument from the command line. This is an optional routine required only if command line processing is implemented. If command line processing is not implemented, implement this routine as a dummy, or remove its call from the code.

INPUT	NUMBER — INTEGER, argument number requested
OUTPUT	STRING — CHARACTER*1 STRING(80), string containing the argument

BCLRED (UNIT, IOS, STATUS)

BCLRED closes a file opened by BOPRED for sequential binary reading.

INPUT	UNIT — INTEGER, the unit to close
OUTPUT	IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero STATUS — INTEGER, the error status zero — allok non-zero — error closing the unit

BINRED (UNIT, COUNT, BUFFER, IOS, STATUS)

BINRED transfers "COUNT" Fortran INTEGERS from an unformatted file. The file was opened by BOPRED.

INPUT

UNIT — INTEGER, the unit from which to read
COUNT — INTEGER, the number of integer words to transfer

OUTPUT

BUFFER — INTEGER BUFFER(COUNT), the buffer of integer words read in
IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status
zero — allok
non-zero — error reading from the unit

BOFRED (UNIT, FNUM, IOS, STATUS)

BOFRED opens a binary fontcap file for sequential unformatted reading. The file was created by the fontcap preprocessor (file FONTC) using the local routine BINOPN.

The example BOFRED, which is included in file TRNSPPRT on the distribution tape is written to open a local file FONT*n* (where *n* is the number in FNUM). For example, a CALL BOFRED(IUNIT,23,IOS,STATUS) would open file named FONT23.

INPUT

UNIT — INTEGER, the unit to open
FNUM — INTEGER, a font number

OUTPUT

IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status
zero — allok
non-zero — error opening the unit

BOPRED (UNIT, FLNAME, IOS, STATUS)

BOPRED opens a binary graphcap file for sequential unformatted reading. The file was created by the graphcap preprocessor using the local routine BINOPN.

INPUT

UNIT — INTEGER, the unit to open. (On UNIX systems using C I/O, this will be an output argument.)
FLNAME — CHARACTER*1, a string containing a pathname, blank terminated

OUTPUT

IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status
zero — allok
non-zero — error opening the unit

FLUS (BUFFER, COUNT, UNIT)

FLUS flushes device-dependent instructions to the output device.

INPUT — BUFFER — INTEGER, a packed array of device instructions
COUNT — INTEGER, the number of 8-bit bytes to transmit
UNIT — INTEGER, the unit where instructions are sent

FRPRMP (BUFFER, COUNT, IOS, STATUS)

FRPRMP sends the USER_PROMPT and waits for response. This routine is used only if the graphcap keyword DEVICE_BATCH is FALSE. Any user response will cause execution to resume.

INPUT — BUFFER — INTEGER, an array containing the user prompt. Unpacked with the low order 8 bits of every word containing the ASCII decimal equivalent of an ASCII character.
COUNT — INTEGER, the number of integer words in BUFFER

OUTPUT — IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status.
zero — allok
non-zero — error writing user prompt, or reading user response

INTEGER FUNCTION IARGCT (DUMMY)

This function is used only when optional command line processing is installed. This routine returns as the function result the number of arguments on the command line. This routine will be used in conjunction with ARGGET.

INPUT — None — DUMMY is a dummy argument

OUTPUT — As the function result, the number of arguments on the command line at translator invocation

INTHDL

INTHDL is provided as a means to handle user interrupts. It is invoked as the first executable statement in CGMTRANS. Basically, this routine should specify what to do in the case of intercepting a user interrupt. If a user starts plotting a complex plot, it is useful to provide some quick exit from plotting if the user does not want to continue plotting and wait several seconds or minutes for the given plot to finish. The implementation of this subroutine is highly machine-specific. The file TRNSPPRT on the distribution tape contains an example INTHDL for standard UNIX systems. This subroutine intercepts user interrupts, flushes the output buffers and returns to text mode. To reproduce this logic on other systems, it is necessary to establish where to go on intercepting a user interrupt, and then when an interrupt is intercepted, call the CGMTRANS subroutines STRTUP, ERASE, ENDIT, and BFDMP.

If you do not want to take the time to implement INTHDL initially, then you should simply supply a dummy subroutine for INTHDL.

MSSG (UNIT, MSGNUM, ADDINF)

MSSG puts out a message on the error unit. The message should read:

Message Flag= MSGNUM Other info= ADDINF

This corresponds to the message format given under the "Status Messages" heading, which is at the end of this manual section.

INPUT — UNIT — INTEGER, the unit number to use for writing the message
MSGNUM — INTEGER, the error number
ADDINF — INTEGER, usually the I/O status, but may be some other information

OUTPUT — None

MTOPEN (UNIT, FLNAME, IOS, STATUS)

MTOPEN opens a file for direct unformatted reading of 1440 byte records. This routine opens a pre-CGM NCAR metafile, or an NCAR CGM.

INPUT — UNIT — INTEGER, the unit to open. (This may be an output argument on UNIX systems using C I/O.)
FLNAME — CHARACTER*1, a string containing the pathname of the metafile

OUTPUT — IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status
zero — allok
non-zero — error opening the file

READIT (BUFFER, MAX)

READIT reads a character string from standard input and puts it into a FORTRAN 77 CHARACTER*1 array.

INPUT — MAX — INTEGER, the maximum number of characters to read into the BUFFER

OUTPUT — BUFFER — CHARACTER*1, array to hold the characters read from standard input

RECRED (UNIT, RECNUM, BUFFER, WORDS, IOS, STATUS)

RECRED reads in a metafile record using direct access I/O. The metafile is opened by the MTOPEN routine.

INPUT — UNIT — INTEGER, the unit number from which to read
RECNUM — INTEGER, the current metafile record to read. Count starts at 1. The metafile records are 1440 bytes long.
WORDS — INTEGER, the number of words to read so that 1440 bytes are transferred

OUTPUT — BUFFER — INTEGER, the array containing the current metafile record
IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status.
- zero — allok
- 1 — error in reading
- 2 — end of file found

WRITIT (STRING, COUNT)

WRITIT writes a character string to standard output.

INPUT — STRING — CHARACTER*1, array of characters to write
COUNT — INTEGER, number of characters to write

OUTPUT — None

Routines Required by Fontcap and Graphcap Processors

This part describes the installation-dependent routines required by the fontcap and graphcap preprocessors. All variable types are given in terms of FORTRAN 77.

The fontcap preprocessor requires GBYTES and SBYTES, which are not discussed here. A description of GBYTES and SBYTES is in Appendix A of Section II of this manual.

BINCLS (UNIT, IOS, STATUS)

BINCLS closes a file opened for sequential binary writing. The file was opened by BINOPN.

INPUT — UNIT — INTEGER, the unit to close

OUTPUT — IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status
- zero — allok
- non-zero — error closing the unit

BINOPN (UNIT, FLNAME, IOS, STATUS)

BINOPN opens a file for sequential binary writing.

INPUT — UNIT — INTEGER, the unit to open. (This may be an output argument on UNIX systems using C I/O.)
FLNAME — CHARACTER*1, string containing the path name of the file to open. The string must be terminated with a blank.

OUTPUT — IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status
zero — allok
non-zero — error opening the file

BINWRI (UNIT, COUNT, BUFFER, IOS, STATUS)

BINWRI transfers the contents of BUFFER to the file named by UNIT. Use a sequential unformatted write to perform the transfer. The file was opened by BINOPN.

INPUT — UNIT — INTEGER, the unit on which to write
COUNT — INTEGER, the number of Fortran integers to transfer from BUFFER
BUFFER — INTEGER, array of data which is to be transferred

OUTPUT — IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status
zero — allok
non-zero — error writing

CHRCLS (UNIT, IOS, STATUS)

CHRCLS closes a file opened for sequential character reading. The file was opened by CHROPN.

INPUT — UNIT — INTEGER, the unit to close

OUTPUT — IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status
zero — allok
non-zero — error closing the unit

CHROPN (UNIT, FLNAME, IOS, STATUS)

CHROPN opens a file for sequential unformatted reads. The file contains characters with a maximum of 80 characters per row (record).

INPUT — FLNAME — CHARACTER*1, string containing the path name of the file to open. The string must be terminated with a blank.

OUTPUT — UNIT — INTEGER, the unit the file was opened on
IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero
STATUS — INTEGER, the error status
- zero — allok
- non-zero — error opening the file

CHRRED (UNIT, COUNT, BUFFER, IPTR, IOS, STATUS)

CHHRED transfers from a character file to a Fortran character array. The file was opened by CHROPN.

INPUT — UNIT — INTEGER, the unit from which to read
COUNT — INTEGER, the number of characters to transfer

OUTPUT — BUFFER — CHARACTER*1, string containing the current row of the file
IPTR — INTEGER, index of the first valid character in BUFFER. (This should always be 1 in a Fortran implementation.)
IOS — INTEGER, the I/O status, valid only if STATUS is set to non-zero. If STATUS is zero, then IOS is set to the number of bytes transferred.
STATUS — INTEGER, the error status
- zero — allok
- one — end of file
- > 1 — system dependent error flag

LaserJet Drivers

This section describes the implementation of the LaserJet drivers that are new with Version 3.00. The LaserJet drivers are functional only with the Fortran version of the NCAR CGM interpreter (sometimes called "ftrans" or "cgmtrans"). The LaserJet drivers will not work with the C version of the NCAR CGM interpreter (sometimes called "ctrans").

There are ten supported graphcaps on the distribution tape for use with HP LaserJet printers. The LaserJets will accept only bit-map images, so the translator has to rasterize its output before sending it to a LaserJet printer. This rasterization process can be quite time consuming. Consequently, there are several different graphcaps to produce plots at various resolutions ranging from 75 dots per inch to 300 dots per inch. Also, there are two graphcaps to support the original LaserJet printers. These are at 75 dots per inch resolution only.

There is a COMMON block in the translator for storage of the output raster image for the LaserJet printers. This is the common block with name RASBUF. Since the storage requirements may be significant, no storage is allocated for the LaserJet raster images by default — the reason being that unless someone is going to use the LaserJet drivers, then a storage penality should not be exacted. If you are going to use any LaserJet printer, you will have to dimension the RBUF array appropriately by redefining MXRBUF.

We now describe how to figure out how big MXRBUF should be. A full image on a LaserJet is plotted on an 8-inch by 10-inch rectangle, with the actual plot being in an 8-inch by 8-inch square to satisfy the aspect ratio requirements of the standards. To figure out how much space needs to be allocated in the RASBUF common array RBUF, take the resolution in dots per inch and first figure out how many bits are needed. For example, if the resolution is 75 dots per inch, then the required number of bits to store the output image is 75 x 8 x 75 x 10 = 450000. To get the required dimension for RBUF, divide the number of bits by the number of bits per word. For the previous example of 75 dots per inch resolution, on a 32-bit machine RBUF should be dimensioned for 450000/32 = 14063. For the maximum resolution of 300 dots per inch on a 32-bit machine, dimension RBUF for 225000. The dimension of RBUF can be changed by using a text editor on the file CGMTRANS to change the value "MXRBUF=1" to whatever it should be. If you want to be assured that you can use any of the LaserJet drivers on a 32-bit machine, set MXRBUF=225000.

Available Graphcaps for LaserJet Printers

hpljo75l	This graphcap supports the original LaserJet family of printers. The only resolution available for those printers is 75 dots per inch. This graphcap produces plots in landscape mode — the X-axis of the plot is parallel with the long side of the page.
hpljo75p	This graphcap supports the original LaserJet family of printers. The only resolution available for those printers is 75 dots per inch. This graphcap produces plots in portrait mode — the X-axis of the plot is parallel with the short side of the page.
hplj75l	This graphcap supports the HP LaserJet, the LaserJet Plus, the LaserJet Series II, the LaserJet 500 Plus, and so on. This driver produces plots in landscape mode at 75 dots per inch resolution.
hplj75p	This graphcap supports the HP LaserJet, the LaserJet Plus, the LaserJet Series II, the LaserJet 500 Plus, and so on. This driver produces plots in portrait mode at 75 dots per inch resolution.
hplj100l	This graphcap supports the HP LaserJet, the LaserJet Plus, the LaserJet Series II, the LaserJet 500 Plus, and so on. This driver produces plots in landscape mode at 100 dots per inch resolution.
hplj100p	This graphcap supports the HP LaserJet, the LaserJet Plus, the LaserJet Series II, the LaserJet 500 Plus, and so on. This driver produces plots in portrait mode at 100 dots per inch resolution.
hplj150l	This graphcap supports the HP LaserJet, the LaserJet Plus, the LaserJet Series II, the LaserJet 500 Plus, and so on. This driver produces plots in landscape mode at 150 dots per inch resolution.
hplj150p	This graphcap supports the HP LaserJet, the LaserJet Plus, the LaserJet Series II, the LaserJet 500 Plus, and so on. This driver produces plots in portrait mode at 150 dots per inch resolution.
hplj300l	This graphcap supports the HP LaserJet, the LaserJet Plus, the LaserJet Series II, the LaserJet 500 Plus, and so on. This driver produces plots in landscape mode at 300 dots per inch resolution.
hplj300p	This graphcap supports the HP LaserJet, the LaserJet Plus, the LaserJet Series II, the LaserJet 500 Plus, and so on. This driver produces plots in portrait mode at 300 dots per inch resolution.

The translator writes its output raster buffer for the LaserJet printers in the same way it writes to any device. If your LaserJet is available on a UNIX system via the "lpr" command, then piping the output of the translator to "lpr" should work. In most instances it will not work just to stream the output from the translator directly to the printer since the printer

needs to be initialized. In the cases where the operating system does not provide an access path to the LaserJet, you will want to capture the output file from the translator and then dispose that file to the LaserJet in a separate command. For a particular example, if you are running on a Sun workstation and a LaserJet is attached to Serial Port A (consult your LaserJet Printer Owner's Reference Manual for pin assignments), then you can dispose the output file from the translator by executing the object module obtained from compiling and loading the following C code (specify the output file from the translator as coming from standard input):

```
#include <stdio.h>
#include <sgtty.h>

main()
{
        int     fd;
        int     c;
        struct sgttyb    control;
        char    *s;
        FILE    *laser;
        int     mode;

        if ( (fd = open("/dev/ttya", 2)) == -1)
        {
                fprintf(stderr, "error opening /dev/ttya0);
                exit(-1);
        }

        ioctl(fd, TIOCGETP, &control);

        control.sg_ispeed = B9600;
        control.sg_ospeed = B9600;
        control.sg_flags &= (~ EVENP );
        control.sg_flags &= (~ ODDP );
        control.sg_flags |= RAW;

        ioctl(fd, TIOCSETP, &control);

        laser = fdopen(fd, "w");

        while ( (c = getchar()) != EOF ) putc(c, laser);
}
```

Status Messages

This section describes the processing status of the metafile translator. This information is displayed on your device at translator termination. The messages have the form:

MESSAGE FLAG=X, OTHER INFO=Y

where X is the translator error number. The errors are listed by number below.

1. Normal termination flag. The OTHER INFO is the number of frames processed. No Error.
2. Error opening the metafile. The OTHER INFO is a system dependent error flag indicating the reason for failure.
3. Error decoding the record position field from the command line. The record pointer given to the translator for display of a specific frame was not a valid integer.
4. Error reading a metafile record. The OTHER INFO will be a system-dependent error status.
5. Metafile type change. This will occur only if a pre-CGM NCAR metafile is mixed with an NCAR CGM.
6. An invalid metafile type was detected. The translator attempts to skip and look for another record to use.
7. Invalid metafile instruction. The translator attempts to continue processing. The Element Class is displayed in OTHER INFO.
8. The record type changed when an instruction crossed a record boundary. The translator will abort.
9. The requested frame has been plotted. Not used when in "display all frame" mode. This should not be displayed.
10. Error encoding an integer value. Typically the integer is larger than allowed by the given field length.
11. The device coordinate data type had an invalid value.
12. Error opening the graphcap file. The OTHER INFO will display a system-dependent status value.
13. Error reading the graphcap file. The OTHER INFO will display a system-dependent status value.
14. Error closing the graphcap file. The OTHER INFO will display a system-dependent status value.
15. Error opening the fontcap file. The OTHER INFO will display the requested font index.
16. Error reading the fontcap file. The OTHER INFO will display the requested font index.
17. Error closing the fontcap file. The OTHER INFO will display the requested font index.
18. Polyline bundle index greater than defined. No action taken.

19. Polymarker bundle index greater than defined. No action taken.
20. Text bundle index greater than defined. No action taken.
21. Polygon bundle index greater than defined. No action taken.
22. Invalid polymarker type requested. No action taken.
23. Error in the number of cell array cells given.
24. CGM color table attribute index larger than table allows. No action taken.
25. No room for latest direct color specification. No action taken.
26. An unknown pre-CGM NCAR metafile instruction was found.
27. An unknown pre-CGM NCAR option instruction setting was found.
28. End of Metafile. Instruction causes the translator to terminate.
29. Metafile and Translator versions do not match. The OTHER INFO is the metafile version number.
30. Bad record size. The record bit count is not in the valid range. The OTHER INFO is the bit count of the bad record.
31. Error writing to the display surface.
32. No version number found. A METAFILE VERSION instruction must be encountered before a BEGIN PICTURE instruction.
33. Divide by zero. An individual fontcap entry has caused a floating point error in subroutine CTRANS — check the fontcap.
34. The number of points in a POLYGON instruction exceeds the maximum allowable for software simulation; the polygon boundary is drawn. The OTHER INFO is the number of points requested. This is a non-fatal message.
35. Invalid FILL AREA style index. The OTHER INFO is the requested index. A boundary is drawn. This is a non-fatal condition.
36. Invalid HATCH INDEX requested. The OTHER INFO is the requested index. This is a non-fatal condition — hatch index 1 is used.
37. Invalid virtual device window specified.
38. Invalid device coordinate space requested.
39. Maximum number of error message number 24's (as described above) have been issued for this frame.
40. Size of the raster buffer for the HP LaserJet driver is not large enough. The OTHER INFO is the dimension of the raster buffer array. See the implementation instructions for the HP LaserJet driver for details.

Appendix A: Translator ASCII Codes

The table that begins below and continues on the following pages describes the ASCII character representations used in the graphcaps. Each character is represented by the code used in graphcaps, its ASCII name, and its decimal position in the collating sequence.

Translator ASCII Codes

CHARACTER	NAME	DECIMAL
NUL	Null	0
SOH	Start of Heading	1
STX	Start of Text	2
ETX	End of Text	3
EOT	End of Transmission	4
ENQ	Enquiry	5
ACK	Acknowledgment	6
BEL	Bell	7
BS	Backspace	8
HT	Horizontal Tab	9
LF	Line Feed	10
VT	Vertical Tab	11
FF	Form Feed	12
CR	Carriage Return	13
SO	Shift Out	14
SI	Shift In	15
DLE	Data Link Escape	16
DC1	Device Control	17
DC2	Device Control	18
DC3	Device Control	19
DC4	Device Control	20
NAK	Negative Acknowledgment	21
SYN	Synchronization Character	22
ETB	End of Transmission Block	23
CAN	Cancel	24
EM	End of Medium	25
SUB	Substitute	26
ESC	Escape	27
FS	Field Separator	28
GS	Group Separator	29
RS	Record Separator	30
US	Unit Separator	31

(continued on next page)

Translator ASCII Codes

CHARACTER	NAME	DECIMAL
SPC	space	32
!	!	33
"	"	34
#	#	35
$	$	36
%	%	37
&	&	38
'	'	39
(	(	40
)	)	41
*	*	42
+	+	43
,	,	44
-	-	45
.	.	46
/	/	47
0	0	48
1	1	49
2	2	50
3	3	51
4	4	52
5	5	53
6	6	54
7	7	55
8	8	56
9	9	57
:	:	58
;	;	59
<	<	60
=	=	61
>	>	62
?	?	63

(continued on next page)

Translator ASCII Codes		
CHARACTER	NAME	DECIMAL
@	@	64
A	A	65
B	B	66
C	C	67
D	D	68
E	E	69
F	F	70
G	G	71
H	H	72
I	I	73
J	J	74
K	K	75
L	L	76
M	M	77
N	N	78
O	O	79
P	P	80
Q	Q	81
R	R	82
S	S	83
T	T	84
U	U	85
V	V	86
W	W	87
X	X	88
Y	Y	89
Z	Z	90
[	[	91
\	\	92
]	]	93
^	^	94
_	_	95

(continued on next page)

Translator ASCII Codes		
CHARACTER	NAME	DECIMAL
`	`	96
a	a	97
b	b	98
c	c	99
d	d	100
e	e	101
f	f	102
g	g	103
h	h	104
i	i	105
j	j	106
k	k	107
l	l	108
m	m	109
n	n	110
o	o	111
p	p	112
q	q	113
r	r	114
s	s	115
t	t	116
u	u	117
v	v	118
w	w	119
x	x	120
y	y	121
z	z	122
{	{	123
|	|	124
}	}	125
~	~	126
DEL	delete	127

Section V: NCAR Computer Graphics Metafile

Section V: NCAR Computer Graphics Metafile

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 material in the section of this manual titled the "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 is a private encoding of 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 broken 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 pre-CGM NCAR metafile.

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 4 bits in the control field comprise the datatype identifier field. Defined datatypes include header (0100 binary), NCAR-formatted printer (1000 binary), pre-CGM NCAR metafile (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 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 metafiles. 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:

Bits	Description
1-16	Byte count for the record
17-20	Data-type flag
21	New-frame bit
22	Begin metafile bit
23	End metafile bit
24-32	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 October 1989. "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 END METAFILE BEGIN PICTURE BEGIN PICTURE BODY END PICTURE	
1	METAFILE VERSION METAFILE DESCRIPTION METAFILE ELEMENT LIST METAFILE DEFAULTS REPLACEMENT FONT LIST	VDC TYPE INTEGER PRECISION REAL PRECISION INDEX PRECISION COLOUR PRECISION COLOUR INDEX PRECISION MAXIMUM COLOUR INDEX COLOUR VALUE EXTENT CHARACTER SET LIST CHARACTER CODING ANNOUNCER
2	COLOUR SELECTION MODE VDC EXTENT BACKGROUND COLOUR	SCALING MODE LINE WIDTH SPECIFICATION MODE MARKER SIZE SPECIFICATION MODE EDGE WIDTH SPECIFICATION MODE
3	VDC INTEGER PRECISION CLIP RECTANGLE CLIP INDICATOR	VDC REAL PRECISION AUXILIARY COLOUR TRANSPARENCY
4	POLYLINE POLYMARKER TEXT POLYGON CELL ARRAY GENERALIZED DRAWING PRIMITIVE	DISJOINT POLYLINE RESTRICTED TEXT APPEND TEXT POLYGON SET RECTANGLE CIRCLE CIRCULAR ARC 3 POINT CIRCULAR ARC 3 POINT CLOSE CIRCULAR ARC CENTRE CIRCULAR ARC CENTRE CLOSE ELLIPSE ELLIPTICAL ARC ELLIPTICAL ARC CLOSE

Class	Supported Elements	Unsupported Elements
5	LINE BUNDLE INDEX LINE TYPE LINE WIDTH LINE COLOUR MARKER BUNDLE INDEX MARKER TYPE MARKER SIZE MARKER COLOUR TEXT BUNDLE INDEX TEXT FONT INDEX TEXT PRECISION CHARACTER EXPANSION FACTOR CHARACTER SPACING TEXT COLOUR CHARACTER HEIGHT CHARACTER ORIENTATION TEXT PATH TEXT ALIGNMENT FILL BUNDLE INDEX INTERIOR STYLE FILL COLOUR HATCH INDEX PATTERN INDEX FILL REFERENCE POINT COLOUR TABLE ASPECT SOURCE FLAGS	CHARACTER SET INDEX ALTERNATE CHARACTER SET INDEX EDGE BUNDLE INDEX EDGE TYPE EDGE WIDTH EDGE COLOUR EDGE VISIBILITY PATTERN TABLE PATTERN SIZE
6	ESCAPE	
7		MESSAGE APPLICATION DATA