NCAR Graphics
Installer’s Guide
Version 2.00  August 1987

Fred Clare
NCAR TECHNICAL NOTES

The Technical Note series provides an outlet for a variety of NCAR manuscripts that contribute in specialized ways to the body of scientific knowledge but which are not suitable for journal, monograph, or book publication. Reports in this series are issued by the NCAR Scientific Divisions. Designation symbols for the series include:

- **EDD** - *Engineering, Design, or Development Reports*
  Equipment descriptions, test results, instrumentation, and operating and maintenance manuals.

- **IA** - *Instructional Aids*
  Instruction manuals, bibliographies, film supplements, and other research or instructional aids.

- **PPR** - *Program Progress Reports*
  Field program reports, interim and working reports, survey reports, and plans for experiments.

- **PROC** - *Proceedings*
  Documentation of symposia, colloquia, conferences, workshops, and lectures. (Distribution may be limited to attendees.)

- **STR** - *Scientific and Technical Reports*
  Data compilations, theoretical and numerical investigations, and experimental results.

Copyright © 1987, University Corporation for Atmospheric Research. All Rights Reserved.

Published by: National Center for Atmospheric Research
Scientific Computing Division
P.O. Box 3000
Boulder, CO 80307

Additional copies of this manual are available from NCAR Graphics Distribution at the above address. Please contact NCAR Graphics Distribution for price information.

The National Center for Atmospheric Research is operated by the University Corporation for Atmospheric Research and is sponsored by the National Science Foundation. Any opinion, findings, conclusions, or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
# CONTENTS

*Preface* ........................................................................................................... v

## Section I: Introduction

- *Introduction* ................................................................................................ 3
- NCAR Graphics Site Representatives ............................................................ 3
- Installation Consulting .................................................................................... 3

## Section II: NCAR Graphics Package Installation Directions

- *Introduction* ................................................................................................. 9
- Package Contents .......................................................................................... 9
- Implementing the System .............................................................................. 9
- Partial Implementations ................................................................................ 9
- Step 1 — Implementation of Lower-level Support Routines ....................... 10
- Step 2 — Implementation of NCAR's GKS 0A Package ............................... 11
- Step 3 — Implementation of SPPS .............................................................. 12
- Step 4 — Implementation of the NCAR CGM Translator ............................ 12
- Step 5 — Implementation of the Higher-level Utilities ............................... 13
- User Libraries ............................................................................................... 14
- Size Considerations ..................................................................................... 14
- Appendix A — Machine-dependent Support Routines ............................... 17
- Appendix B — Dumps of the Metafile from TGKSOA ............................... 23

## Section III: Release Contents

- *Introduction* ................................................................................................. 35
- NCAR GKS 0A .............................................................................................. 35
- Required Support Routines ........................................................................... 35
- Implementation Tools ................................................................................... 36
- Utilities ........................................................................................................ 36
- Utility Support Files ..................................................................................... 38
- CGM Translator ........................................................................................... 40
- Data Bases .................................................................................................... 41
- Data Base Initializers ................................................................................... 42
- Bookkeeping ................................................................................................. 42
Section IV: NCAR Computer Graphics Metafile
Translator

### Introduction ........................................ ................................................ 47

- Background.................................................................................................. 49
- The Metafile Standard ........................................................................... 50
- The NCAR CGM Translator ..................................................................... 51
- Installation ................................................................................................. 53

### GRAPHCAP Files .............................................................................. 57

- Introduction................................................................................................ 59
- Formatting and Encoding Coordinates................................................... 61
- Workstation Initializations ....................................................................... 64
- Workstation Drawing Space ..................................................................... 65
- Device Vector Counts ............................................................................ 68
- Device Color Capabilities ....................................................................... 69
- Line Control ............................................................................................... 73
- Marker Control .......................................................................................... 77
- Graphical Text Control ............................................................................ 77
- Bundle Tables ............................................................................................. 78
- Polygon Control ......................................................................................... 81
- Raster Control ............................................................................................ 82

### FONTCAP Files .................................................................................... 87

- Introduction ................................................................................................ 89
- Character Class ......................................................................................... 89
- Font Class .................................................................................................. 90
- Coordinate Class ....................................................................................... 91
- Character Stroke Class ............................................................................ 92

### Required Locally-implemented Routines .............................................. 95

- Introduction ................................................................................................ 97
- Routines Required by the Translator ................................................... 97
- Routines Required by FONTCAP and GRAPHCAP Processors ........... 101

### Status Messages ................................................................................... 105

### Appendix A — Translator ASCII Codes ............................................. 109

Section V: NCAR Computer Graphics Metafile

### Introduction ......................................................................................... 117

- Record Formatting and NCAR Datatypes .............................................. 117
- Supported and Unsupported Elements ..................................................... 119

NCAR Graphics Site Representative Information Update Form ............. 123
Preface

This NCAR Graphics Installer's Guide is version 2.00. It is compatible with version 2.00 of the NCAR Graphics Package code, distribution of which began in October 1987. Please read the Introduction section of this Guide for more details.

The author would like to acknowledge John Humbrecht as the author of the original draft of The NCAR Computer Graphics Metafile Translator. Thanks go to David Kennison, Bob Lackman, Donna Converse, and Peter Fox for their reading through the drafts and making suggestions and error corrections. Thanks go to Nancy Dawson and Ginger Caldwell for their advice on the formatting of the text, and for suggesting the idea of making this a separate manual. Special thanks to Belinda Housewright for preparing several of the tables, and to Belinda, Lisa Benson, and Christine Guzy for their help in preparing the final copy. Also, thanks go to the many users of previous documentation who made valuable criticisms and comments.
SECTION I: Introduction
Introduction


The material in this manual is intended to be used by an experienced programmer in the initial installation of the NCAR Graphics package. It is assumed that the user of this manual will have perused the companion manual: NCAR Graphics User's Guide (NCAR/TN-283+IA), August 1987. This manual makes frequent references to the User's Guide. After reading this introduction, proceed to Section II for installation details.

The NCAR Graphics Installer's Guide has five sections:

1. Introduction. This is the section you are currently reading.

2. NCAR Graphics Package 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 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.

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 NCAR 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, fill out the form that is the last page of this manual, or write to:

National Center for Atmospheric Research
Scientific Computing Division
NCAR Graphics Distribution
P.O. Box 3000
Boulder, CO 80307-3000

Installation Consulting

If you have questions related to installing the package, call NCAR Graphics Installation at (303) 497-1309.
Section II: NCAR Graphics Package
Installation Directions
<table>
<thead>
<tr>
<th>Section II: NCAR Graphics Package Installation Directions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction ........................................................................................................... 9</td>
</tr>
<tr>
<td>Package Contents .................................................................................................... 9</td>
</tr>
<tr>
<td>Implementing the System ...................................................................................... 9</td>
</tr>
<tr>
<td>Partial Implementations ....................................................................................... 9</td>
</tr>
<tr>
<td>Step 1 — Implementation of Lower-level Support Routines ......................... 10</td>
</tr>
<tr>
<td>Step 2 — Implementation of NCAR's GKS 0A Package ................................. 11</td>
</tr>
<tr>
<td>Step 3 — Implementation of SPPS ................................................................. 12</td>
</tr>
<tr>
<td>Step 4 — Implementation of the NCAR CGM Translator .............................. 12</td>
</tr>
<tr>
<td>Step 5 — Implementation of the Higher-level Utilities ................................. 13</td>
</tr>
<tr>
<td>User Libraries .................................................................................................... 14</td>
</tr>
<tr>
<td>Size Considerations ........................................................................................... 14</td>
</tr>
<tr>
<td>Appendix A — Machine-dependent Support Routines ................................... 17</td>
</tr>
<tr>
<td>Appendix B — Dumps of the Metafile from TGKS0A ................................. 23</td>
</tr>
</tbody>
</table>
NCAR Graphics Package Installation Directions

Introduction

This guide is designed to provide step-by-step guidance for the implementation of the NCAR GKS-compatible Graphics System. 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 package on any system.

This manual assumes that you have read the Introduction section in the NCAR Graphics User's Guide (hereafter referred to as the User's Guide.)

This guide describes an implementation procedure that begins at the lowest levels of the system and proceeds to the highest levels.

Package Contents

A detailed listing of the contents of the NCAR GKS-compatible Graphics System is given in Release Contents, which is Section III of this manual. In the cover letter which accompanies the distribution tape is a list of file names in the sequence that the files occur on the 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.

Implementing the System

The major components involved in a full implementation of the NCAR graphics package are listed below.

- Machine-specific support routines.
- A GKS 0A package.
- SPPS — The NCAR System Plot Package Simulator.
- Higher-level Utilities and demonstration drivers.

Each of these components will be 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. It is possible to implement certain components of the package independently, however.

If one is using a commercial GKS package and simply wants to run some, or all, of the higher-level utilities (AUTOGRPH, CONREC, EZMAP, and so on) on top of their 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. If one wants 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 discussions provide 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 in the Examples section of the User's Guide.

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`
- `G01MIO`
- `IAND`
- `IOR`
- `ISHIFT`
- `IIMACH`
- `R1MACH`
- `SBYTES`

These routines are referred to as "low-level support routines" in the remainder of this section. 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. You may want to modify `G01MIO` to allow for user-specification of the metafile output 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.

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 which 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.

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. GKS can be implemented at various levels of complexity. The NCAR implementation is a level 0A implementation. GKS at Level 0A does not allow for picture segmentation or graphical input; GKS at Level 0A has approximately the same functionality as the NCAR System Plot Package in pre-GKS versions of the NCAR Graphics Package. AWI stands for "Above Workstation Interface" and BWI stands for "Below Workstation Interface". 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 sections NCAR Computer Graphics Metafile Translator and NCAR Computer Graphics Metafile in this manual.

If you wanted 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 one loads and executes 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 TGKSOA. 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 desired metafile are provided in Appendix B 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 that contained in the Examples section of the User's Guide. 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.)

On EBCDIC computers (most IBM equipment) the EBCDIC character set must be converted to ASCII prior to being written to the CGM. 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 EBCDIC 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 Examples section of 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 document 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.

For each of the higher-level utilities (those appearing in the *Utilities* section of the User's Guide), 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 Tname
CALL CLSGKS
```

There are four 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.
- **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 those that appear in the *Examples* section of the User's Guide.

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

- A main program which 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.
- 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 *Utilities* section of the User's Guide for any special implementation considerations. For example:

- EZMAP and PWRITX require connecting to support data files.
• ISOSRFHR requires a scratch file.
• DASHSUPR requires some code modification.
• CONRCQCK and CONRAQ use DASHLINE for drawing lines.
• CONREC and CONRAN use DASHCHAR for drawing lines and labels.
• CONRCSPR and CONRAS use DASHSUPR for drawing lines and labels.
• DASHSMTH is used with CONREC, then the contour lines are passed through a smoother.

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.

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). Most implementors solve this problem by simply including the most generic versions of these packages, CONREC and DASHCHAR, into their binary libraries.

4. *User aids.* The routine FINDG should be provided if users will be wanting to convert their applications from using the pre-GKS version of the NCAR package to the current GKS version.

5. *Demo drivers.* Users find that it is very helpful to take the source for the demo drivers and modify it to suit their own particular needs.


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 UNIX™-based Pyramid 90X minicomputer, the object module created for running the CONREC demonstration driver (sources: application driver; the CONREC demo TCONRE; the CONREC source; SPPS; NCAR’s GKS package; and the low-level support routines) is 286720 bytes in size. Different utilities will of course have different space requirements. The binary database used by EZMAP requires 587264 bytes on a Pyramid, 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.

---

TUNIX™ is a trademark of Bell Laboratories.

August 1987

14
## Metafile Sizes

<table>
<thead>
<tr>
<th>Utility</th>
<th>CGM Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTOGRPH</td>
<td>18720</td>
</tr>
<tr>
<td>CONRAN</td>
<td>15844</td>
</tr>
<tr>
<td>CONRAQ</td>
<td>24480</td>
</tr>
<tr>
<td>CONRAS</td>
<td>21600</td>
</tr>
<tr>
<td>CONRCQCK</td>
<td>17280</td>
</tr>
<tr>
<td>CONRCSPR</td>
<td>24480</td>
</tr>
<tr>
<td>CONREC</td>
<td>18720</td>
</tr>
<tr>
<td>DASHCHAR</td>
<td>7200</td>
</tr>
<tr>
<td>DASHLINE</td>
<td>7200</td>
</tr>
<tr>
<td>DASHSMTH</td>
<td>8640</td>
</tr>
<tr>
<td>DASHSUPR</td>
<td>10080</td>
</tr>
<tr>
<td>EZMAP</td>
<td>1344960</td>
</tr>
<tr>
<td>GRIDAL</td>
<td>25920</td>
</tr>
<tr>
<td>HAFTON</td>
<td>491040</td>
</tr>
<tr>
<td>HSTGRM</td>
<td>10080</td>
</tr>
<tr>
<td>ISOSRF</td>
<td>38880</td>
</tr>
<tr>
<td>ISOSRFHR</td>
<td>38880</td>
</tr>
<tr>
<td>PWRITX</td>
<td>93600</td>
</tr>
<tr>
<td>PWRITY</td>
<td>8640</td>
</tr>
<tr>
<td>PWRZI</td>
<td>20160</td>
</tr>
<tr>
<td>PWRZS</td>
<td>17280</td>
</tr>
<tr>
<td>PWRZT</td>
<td>5760</td>
</tr>
<tr>
<td>SURFACE</td>
<td>21600</td>
</tr>
<tr>
<td>STRMLN</td>
<td>14400</td>
</tr>
<tr>
<td>THREED</td>
<td>8640</td>
</tr>
<tr>
<td>VELVCT</td>
<td>28800</td>
</tr>
</tbody>
</table>
Appendix A: Machine-dependent Support Routines
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 his 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:

\[ \text{SIGN} \ast (X(S-1) \ast A^{S-1} + \ldots + X(1) \ast A + X(0)) \]

in which \( 0 < X(I) < A \) for \( I = 0, \ldots, S-1 \).

- 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:

\[ \text{SIGN} \ast (B^E) \ast ((X(1)/B + \ldots + (X(T)/B^T)) \]

in which \( 0 < X(1) \), and EMIN \( \leq E \leq EMAX \).

- I1MACH(10) = B, the base

**Single-precision Constants**
- I1MACH(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:

\[
\begin{align*}
R1MACH(1) &= B^{**(EMIN-1)}, \text{ the smallest positive magnitude} \\
R1MACH(2) &= B^{**EMAX*(1-B**(-T))}, \text{ the largest magnitude} \\
R1MACH(3) &= B^{**(-T)}, \text{ the smallest relative spacing} \\
R1MACH(4) &= B^{**(1-T)}, \text{ the largest relative spacing} \\
R1MACH(5) &= \log_{10}(B)
\end{align*}
\]

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). No assumption is made about what bits are added at the top of the word; these bits remain undefined.

It is required 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, 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 should work correctly on most machines.

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 buffer to IABS(IUNIT).

- **IUNIT** IABS(IUNIT) is the FORTRAN logical unit number on which IOP is to occur.
- **IBUFF** Buffer containing data for a 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 obtains 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 obtains 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
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

<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Bytes dumped in octal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>000 036 064 000 000 041 000 000 020 042 200 001 020 113 012 116</td>
</tr>
<tr>
<td>0017</td>
<td>103 101 122 137 107 113 123 060 101 000 021 146 000 001 377 377</td>
</tr>
<tr>
<td>0033</td>
<td>000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000</td>
</tr>
</tbody>
</table>

The remaining 1392 bytes of record #1 are zero
### Octal Dump — Record #2

<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Bytes dumped in octal</th>
<th>Record #2 (1440 bytes total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>002 054 070 000 000 141 000 000 000 200 060 302 000 001 060 250</td>
<td></td>
</tr>
<tr>
<td>0017</td>
<td>014 314 014 314 163 062 163 062 100 077 000 044 021 353 141 106</td>
<td></td>
</tr>
<tr>
<td>0033</td>
<td>027 012 153 204 034 050 141 106 041 107 153 204 046 146 141 106</td>
<td></td>
</tr>
<tr>
<td>0049</td>
<td>053 204 153 204 060 243 141 106 065 302 153 204 072 340 141 106</td>
<td></td>
</tr>
<tr>
<td>0065</td>
<td>120 302 000 002 100 177 000 044 021 353 107 255 027 012 121 352</td>
<td></td>
</tr>
<tr>
<td>0081</td>
<td>034 050 107 255 041 107 121 352 046 146 107 255 053 204 121 352</td>
<td></td>
</tr>
<tr>
<td>0097</td>
<td>060 243 107 255 065 302 121 352 072 340 107 255 122 302 000 001</td>
<td></td>
</tr>
<tr>
<td>0113</td>
<td>124 050 000 000 105 036 105 036 000 000 123 344 077 377 077 377</td>
<td></td>
</tr>
<tr>
<td>0129</td>
<td>100 377 000 044 137 174 153 117 132 355 155 371 125 301 155 015</td>
<td></td>
</tr>
<tr>
<td>0145</td>
<td>122 140 151 002 122 140 143 311 125 301 137 275 132 355 156 321</td>
<td></td>
</tr>
<tr>
<td>0161</td>
<td>137 174 141 173 141 106 146 145 101 077 001 064 112 074 107 255</td>
<td></td>
</tr>
<tr>
<td>0177</td>
<td>150 364 124 172 150 364 107 255 000 030 000 014 000 000 000 001</td>
<td></td>
</tr>
<tr>
<td>0193</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0209</td>
<td>001 000 001 000 001 000 001 000 001 000 000 000 000 000 000 000</td>
<td></td>
</tr>
<tr>
<td>0225</td>
<td>000 001 000 001 000 001 000 001 000 001 000 001 000 001 000 001</td>
<td></td>
</tr>
<tr>
<td>0241</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0257</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0273</td>
<td>000 001 000 001 000 001 000 001 000 001 000 001 000 001 000 001</td>
<td></td>
</tr>
<tr>
<td>0289</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0305</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0321</td>
<td>000 001 000 001 000 001 000 001 000 001 000 001 000 001 000 001</td>
<td></td>
</tr>
<tr>
<td>0337</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0353</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0369</td>
<td>000 001 000 001 000 001 000 001 000 001 000 001 000 001 000 001</td>
<td></td>
</tr>
<tr>
<td>0385</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0401</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0417</td>
<td>000 001 000 001 000 001 000 001 000 001 000 001 000 001 000 001</td>
<td></td>
</tr>
<tr>
<td>0433</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0449</td>
<td>001 000 001 000 001 000 001 000 001 000 001 000 001 000 001 000</td>
<td></td>
</tr>
<tr>
<td>0465</td>
<td>000 001 000 001 000 001 000 001 000 001 000 001 000 001 000 001</td>
<td></td>
</tr>
<tr>
<td>0481</td>
<td>122 114 000 002 000 003 000 000 000 000 000 000 000 000 000 000</td>
<td></td>
</tr>
<tr>
<td>0497</td>
<td>002 014 122 010 000 000 002 014 02 014 02 014 000 000 010 037 000 055</td>
<td></td>
</tr>
<tr>
<td>0513</td>
<td>077 377 046 146 000 000 046 105 170 141 155 160 154 145 140 163</td>
<td></td>
</tr>
<tr>
<td>0529</td>
<td>164 162 151 156 147 054 040 143 145 156 164 145 162 145 144 040</td>
<td></td>
</tr>
<tr>
<td>0545</td>
<td>151 155 040 164 150 145 040 143 145 156 164 145 162 000 000 240</td>
<td></td>
</tr>
<tr>
<td>0561</td>
<td>000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000</td>
<td></td>
</tr>
</tbody>
</table>

The remaining 864 bytes of record #2 are zero.
### Octal Dump — Record #3

<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Bytes dumped in octal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>000 002 062 000 000 100</td>
</tr>
</tbody>
</table>

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.)

### Hex Dump — Record #1

<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Bytes dumped in hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>001E 3400 0021 0000 1022 8001 104B OA4E</td>
</tr>
<tr>
<td>0017</td>
<td>4341 525F 474B 5330 4100 1166 0001 FFFF</td>
</tr>
<tr>
<td>0033</td>
<td>0000 0000 0000 0000 0000 0000 0000 0000</td>
</tr>
</tbody>
</table>

The remaining 1392 bytes of record #1 are zero.
## Hex Dump — Record #2

<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Bytes dumped in hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>022C 3800 0061 0000 0080 30C2 0001 30A8</td>
</tr>
<tr>
<td>0017</td>
<td>0CCC 0CCC 7332 7332 403F 0024 11EB 6146</td>
</tr>
<tr>
<td>0033</td>
<td>170A 6B84 1C28 6146 2147 6B84 2666 6146</td>
</tr>
<tr>
<td>0049</td>
<td>2B84 6B84 30A3 30A3 6B84 7332 51EA 3AE0</td>
</tr>
<tr>
<td>0065</td>
<td>50C2 0002 407F 0024 11EB 0000 11EB 6146</td>
</tr>
<tr>
<td>0081</td>
<td>1C28 47AD 2147 51EA 2666 47AD 2B84 51EA</td>
</tr>
<tr>
<td>0097</td>
<td>30A3 47AD 35C2 51EA 3AE0 47AD 52C2 0001</td>
</tr>
<tr>
<td>0113</td>
<td>5428 0000 451E 451E 0000 53E4 3FFF 3FFF</td>
</tr>
<tr>
<td>0129</td>
<td>40FF 0024 5F7C 6B4F 5AED 6DF9 55C1 6DD0</td>
</tr>
<tr>
<td>0145</td>
<td>5250 6902 5260 63C9 55C1 59ED 5AED 5ED1</td>
</tr>
<tr>
<td>0161</td>
<td>5F7C 617B 6146 6665 413F 0014 4A3C 47AD</td>
</tr>
<tr>
<td>0177</td>
<td>68F4 547A 68F4 47AD 0018 000C 0000 0001</td>
</tr>
<tr>
<td>0193</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0209</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0225</td>
<td>0001 0001 0001 0001 0001 0001 0001 0001</td>
</tr>
<tr>
<td>0241</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0257</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0273</td>
<td>0001 0001 0001 0001 0001 0001 0001 0001</td>
</tr>
<tr>
<td>0289</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0305</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0321</td>
<td>0001 0001 0001 0001 0001 0001 0001 0001</td>
</tr>
<tr>
<td>0337</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0353</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0369</td>
<td>0001 0001 0001 0001 0001 0001 0001 0001</td>
</tr>
<tr>
<td>0385</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0401</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0417</td>
<td>0001 0001 0001 0001 0001 0001 0001 0001</td>
</tr>
<tr>
<td>0433</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0449</td>
<td>0100 0100 0100 0100 0100 0100 0100 0100</td>
</tr>
<tr>
<td>0465</td>
<td>0001 0001 0001 0001 0001 0001 0001 0001</td>
</tr>
<tr>
<td>0481</td>
<td>524C 0002 0003 0000 0000 0000 0000 0000</td>
</tr>
<tr>
<td>0497</td>
<td>020C 5208 0000 020C 020C 020C 0000 409F</td>
</tr>
<tr>
<td>0513</td>
<td>3FFF 2666 0000 2645 7861 6D70 6C65 2073</td>
</tr>
<tr>
<td>0529</td>
<td>7472 69E6 672C 2063 65E6 7465 7265 6420</td>
</tr>
<tr>
<td>0545</td>
<td>69E6 2074 6865 2063 65E6 7465 7265 00A0</td>
</tr>
<tr>
<td>0561</td>
<td>0000 0000 0000 0000 0000 0000 0000 0000</td>
</tr>
</tbody>
</table>

The remaining 864 bytes of record #2 are zero.
### Hex Dump — Record #3

<table>
<thead>
<tr>
<th>Decimal Address</th>
<th>Bytes dumped in hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0002 3200 0040 ... ... ... 0001</td>
</tr>
</tbody>
</table>

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)
Section III: Release Contents
Section III: Release Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>35</td>
</tr>
<tr>
<td>NCAR GKS 0A</td>
<td>35</td>
</tr>
<tr>
<td>AWi</td>
<td>35</td>
</tr>
<tr>
<td>BWI</td>
<td>35</td>
</tr>
<tr>
<td>Required Support Routines</td>
<td>35</td>
</tr>
<tr>
<td>SUPPORT</td>
<td>35</td>
</tr>
<tr>
<td>LOCAL</td>
<td>35</td>
</tr>
<tr>
<td>Implementation Tools</td>
<td>36</td>
</tr>
<tr>
<td>IMPLTEST</td>
<td>36</td>
</tr>
<tr>
<td>FINDG</td>
<td>36</td>
</tr>
<tr>
<td>Utilities</td>
<td>36</td>
</tr>
<tr>
<td>AREAS</td>
<td>36</td>
</tr>
<tr>
<td>AUTOGRPH</td>
<td>36</td>
</tr>
<tr>
<td>CONRAN</td>
<td>37</td>
</tr>
<tr>
<td>CONRAQ</td>
<td>37</td>
</tr>
<tr>
<td>CONRAS</td>
<td>37</td>
</tr>
<tr>
<td>CONREC</td>
<td>37</td>
</tr>
<tr>
<td>CONRCQCK</td>
<td>37</td>
</tr>
<tr>
<td>CONRCSPR</td>
<td>37</td>
</tr>
<tr>
<td>DASHCHAR</td>
<td>37</td>
</tr>
<tr>
<td>DASHLINE</td>
<td>37</td>
</tr>
<tr>
<td>DASHSMTH</td>
<td>37</td>
</tr>
<tr>
<td>DASHSUPR</td>
<td>37</td>
</tr>
<tr>
<td>EZMAP</td>
<td>37</td>
</tr>
<tr>
<td>EZMAPA</td>
<td>37</td>
</tr>
<tr>
<td>GRIDAL</td>
<td>37</td>
</tr>
<tr>
<td>HAFTON</td>
<td>38</td>
</tr>
<tr>
<td>HISTGR</td>
<td>38</td>
</tr>
<tr>
<td>ISOSRF</td>
<td>38</td>
</tr>
<tr>
<td>ISOSRFHR</td>
<td>38</td>
</tr>
<tr>
<td>PWRITX</td>
<td>38</td>
</tr>
<tr>
<td>PWRITY</td>
<td>38</td>
</tr>
<tr>
<td>PWRZI</td>
<td>38</td>
</tr>
<tr>
<td>PWRZS</td>
<td>38</td>
</tr>
<tr>
<td>PWRZT</td>
<td>38</td>
</tr>
<tr>
<td>SURFACE</td>
<td>38</td>
</tr>
<tr>
<td>STRMLN</td>
<td>38</td>
</tr>
<tr>
<td>THREED</td>
<td>38</td>
</tr>
<tr>
<td>VELVCT</td>
<td>38</td>
</tr>
<tr>
<td>Utility Support Files</td>
<td>38</td>
</tr>
<tr>
<td>AGUPWRTX</td>
<td>38</td>
</tr>
<tr>
<td>BNCCHMK</td>
<td>39</td>
</tr>
<tr>
<td>CONCOM</td>
<td>39</td>
</tr>
<tr>
<td>CONTERP</td>
<td>39</td>
</tr>
<tr>
<td>SPPS</td>
<td>39</td>
</tr>
<tr>
<td>TAREAS</td>
<td>39</td>
</tr>
</tbody>
</table>
CGM Translator
CGMTRANS
TRNSPRT

Data Bases
EZMAPDAT
FONTCAP
GRAPHCAP
PWRITXC1
PWRITXC2
PWRITXD1
PWRITXD2
RANFDAT

Data Base Initializers
FONTC
GRAPHC
PWRITXNT

Bookkeeping
F NAMES
HEADER
Release Contents

Introduction

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

NCAR GKS-OA

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 (see 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.

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

ERPR77: An error handling package, adapted from the nonproprietary part of the PORT Mathematical Subroutine Library from Bell Labs.

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

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 Package Installation
Directions" section of this manual are satisfied.

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

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 argu-
ment, 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 TGKSOA: 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.

TSPPS: A program that extensively tests all SPPS (System Plot Pack-
age Simulator) entries. The code contains octal and hex dumps of the
metafile that should be produced.

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

Utilities

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 re-
ferred to as AUTOGRAPH.
CONRAN  Contours irregularly spaced data, labeling the contour lines. Requires DASHCHAR (or DASHSMTH, if smoothed curves are desired). Also requires CONCOM and CONTERP.

CONRAQ  Like CONRAN, but smaller and faster because it has no labeling capacity. Also requires CONTERP.

CONRAS  Like CONRAN, but bigger and slower because lines are smoothed and crowded lines are removed. Also requires CONCOM and CONTERP.

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

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 CONRCQCK.

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 CONRCSPR.

DASHCHAR  Software dashed line package with labeling capability.

DASHLINE  Like DASHCHAR, but smaller and faster because it has no labeling capability. 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/or world political outlines according to one of ten projections. Requires the continental outline data base 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).

GRIDAL  Package for drawing graph paper, backgrounds, perimeters, and so on.
HAFTON  Halftone (gray scale) pictures from a two-dimensional array.

HISTGR  General purpose package for drawing histograms (bar charts).

ISOSRF  Iso-surfaces (with hidden lines removed) from a three-dimensional array.
         May optionally include PWRZI for drawing characters in three space.

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

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

PWRITY  Draws simple software characters.

PWRZI   To be used with ISOSRF to draw characters in three space.

PWRZS   To be used with SRFACE to draw characters in three space.

PWRZT   To be used with THREED to draw characters in three space.

SRFACE  Three-dimensional display of a surface (with hidden lines removed) from a
two-dimensional array. May optionally include PWRZS for drawing char-
         acters in three space.

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-space line drawing capabilities, with entry points
         equivalent to the line drawing entry points of the System Plot Package
         Simulator. May optionally include PWRZT for drawing characters in
         three space.

VELVCT  Two-dimensional velocity field displayed by drawing arrows from the data
         locations. Requires the continental outline database EZMAPDAT. two-
         dimensional array. May optionally include PWRZS, for drawing charac-
         ters in three space.

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 of overlaying the output from one higher-level utility to another. It calls the following utilities: GRIDAL, AUTOGRPH, CONREC, DASHCHAR, VELVCT, and EZMAP. Additionally, it requires the data base 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 old 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 old 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.

TAREAS
Demonstration and routine for the AREAS utility.

TAUTOG
Demonstration subroutine for the AUTOGRPH utility.

TCNQCK
Demonstration subroutine for the CONRCQCK utility.

TCNSMT
Demonstration subroutine for the CNRCISMTH utility.

TCNSUP
Demonstration subroutine for the CONRCSPR utility.

TCONAN
Demonstration subroutine for the CONRAN utility.

TCONAQ
Demonstration subroutine for the CONRAQ utility.

TCONAS
Demonstration subroutine for the CONRAS 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.
NCAR Graphics Installer's Guide

TEZMAP  Demonstration subroutine for the EZMAP utility.
TEZMPA  Demonstration subroutine for the EZMAPA 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.
TPWRTX  Demonstration subroutine for the PWRTIX utility.
TPWRY   Demonstration subroutine for the PWRY utility.
TPWRZI  Demonstration subroutine for the PWRZI utility.
TPWRZS  Demonstration subroutine for the PWRZS utility.
TPWRZT  Demonstration subroutine for the PWRZT 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.

BOPRED: Open a file for sequential unformatted reading. The file was created either by the GRAPHCAP or FONTCAP preprocessors 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.

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 OA 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.

Data Bases

EZMAPDAT Continental outlines data base for the EZMAP utility.

FONTCAP Master file of all FONTCAPs for the CGM Translator. In ASCII form, it must be run through the FONTC preprocessor (below) to convert it to a binary form, which the CGM Translator expects.

GRAPHCAP Master file of all GRAPHCAPs for the CGM Translator.

PWrittxc1 Part I of card image representation of PWrittX Complex font digitization. Must be run through the PWrittXNT preprocessor to convert it to a binary form, which the PWrittX utility expects.
NCAR Graphics Installer's Guide

**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 from the BNCHMK file.

**Data Base Initializers**

**FONTC**  Preprocessor for the FONTCAP data base 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 data base 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 data base files PWRITXC1, PWRITXC2, PWRITXD1, and PWRITXD2 to convert them into binary PWRITX data base.

**Bookkeeping**

**FAMES**  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.
Section IV: NCAR Computer Graphics
Metafile Translator
**Section IV: NCAR Computer Graphics Metafile Translator**

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>47</td>
</tr>
<tr>
<td>Background</td>
<td>49</td>
</tr>
<tr>
<td>The Metafile Standard</td>
<td>50</td>
</tr>
<tr>
<td>The NCAR CGM Translator</td>
<td>51</td>
</tr>
<tr>
<td>Installation</td>
<td>53</td>
</tr>
<tr>
<td>GRAPHCAP Files</td>
<td>57</td>
</tr>
<tr>
<td>Introduction</td>
<td>59</td>
</tr>
<tr>
<td>Formatting and Encoding Coordinates</td>
<td>61</td>
</tr>
<tr>
<td>The Formatting Process</td>
<td>61</td>
</tr>
<tr>
<td>The Encoding Process</td>
<td>63</td>
</tr>
<tr>
<td>Workstation Initializations</td>
<td>64</td>
</tr>
<tr>
<td>Workstation Drawing Space</td>
<td>65</td>
</tr>
<tr>
<td>Device Vector Counts</td>
<td>68</td>
</tr>
<tr>
<td>Device Color Capabilities</td>
<td>69</td>
</tr>
<tr>
<td>Line Control</td>
<td>73</td>
</tr>
<tr>
<td>Marker Control</td>
<td>77</td>
</tr>
<tr>
<td>Graphical Text Control</td>
<td>77</td>
</tr>
<tr>
<td>Bundle Tables</td>
<td>78</td>
</tr>
<tr>
<td>Polygon Control</td>
<td>81</td>
</tr>
<tr>
<td>Raster Control</td>
<td>82</td>
</tr>
<tr>
<td>FONTCAP Files</td>
<td>87</td>
</tr>
<tr>
<td>Introduction</td>
<td>89</td>
</tr>
<tr>
<td>Character Class</td>
<td>89</td>
</tr>
<tr>
<td>Font Class</td>
<td>90</td>
</tr>
<tr>
<td>Coordinate Class</td>
<td>91</td>
</tr>
<tr>
<td>Character Stroke Class</td>
<td>92</td>
</tr>
<tr>
<td>Required Locally-implemented Routines</td>
<td>95</td>
</tr>
<tr>
<td>Introduction</td>
<td>97</td>
</tr>
<tr>
<td>Routines Required by the Translator</td>
<td>97</td>
</tr>
<tr>
<td>Routines Required by FONTCAP and GRAPHCAP Processors</td>
<td>101</td>
</tr>
<tr>
<td>Status Messages</td>
<td>105</td>
</tr>
<tr>
<td>Appendix A — Translator ASCII Codes</td>
<td>109</td>
</tr>
</tbody>
</table>
Introduction
BACKGROUND

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

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

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

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

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

There are two primary drawbacks with having a metafile step between the picture generation and viewing. One drawback is overhead — given a host/plotting device pair, it would be more efficient if the host generated the device instructions for the given device directly and displayed the result. In exchange for simplicity and flexibility, we have compromised efficiency. Also, metafiles are limited in interactive work — 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 contained representatives from several dozen organizations, including almost all of the major computing vendors and government labs. The CGM has also been accepted as an ISO (International Standards Organization) standard.

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

The NCAR GKS-compatible Graphics System creates, on output, a CGM file (conforming to the NCAR-specific record structure). To plot the encoded pictures in an NCAR CGM, the Scientific Computing Division of NCAR 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 current NCAR metafile translator is capable of accepting either the old NCAR metacode (now officially known as pre-CGM NCAR metacode), or NCAR CGM files. The GKS feature of Pattern filling is not supported in this version of the NCAR CGM translator. Polygon style of other than hollow is only supported on devices that provide hardware fill capabilities. Later versions of the translator will include these features. All other features which may be generated by the NCAR GKS OA 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. Complete documentation for implementing the translator on devices having a software interface will appear in future editions of this manual. 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 on-line, one needs 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 GRAPHCAP 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. Currently, about two dozen GRAPHCAPs are supplied on the distribution tape, and only one FONTCAP is supplied. A schematic of how the translator works is given in Figure 1 below. 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.

The GRAPHCAP concept is designed to make it easy to bring on-line 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.

![Figure 1](image-url)
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 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 requires the following 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. They deal with I/O and bit manipulation. Some of the routines are the same as those required by the NCAR GKS OA 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. Two reads are done from FORTRAN logical unit 5 — the first requests the name of the ASCII FONTCAP file to be used as input, and the second requests a name to be given to the binary FONTCAP file which is produced on output. The only FONTCAP file distributed is in file FONTCAP. Plans exist for adding many new fonts in future releases. To create new 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. Two reads are done from FORTRAN logical unit 5 — the first requesting a name for an ASCII GRAPHCAP file as input, and the second requesting a name for the binary GRAPHCAP file produced as output. Over two dozen ASCII 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 the line "/* #####". Since the GRAPHCAP preprocessor regards any line beginning with the two-character sequence "/*" as a comment line, it is optional whether one retains the initial line in the GRAPHCAP for a particular device. If a GRAPHCAP does not exist in GRAPHCAP for a device you want to bring on-line, 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 GRCPN, and this is initialized in BLOCKDATA TRNDAT by the statement

```
DATA (GRCPN(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 one typically wants 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.

- Hard-wire the GRAPHCAP filename by redefining the DATA statement in BLOCKDATA 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 pathname 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.

August 1987
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.

This situation is analogous to the situation with the GRAPHCAPs as described above. The FONTCAP file is read by the translator in subroutine REDFNT. In the default state, the FONTCAP filename is stored in the CHARACTER*1 array FNCAPN, and this is initialized in BLOCKDATA TRNDAT by the statement

```
DATA (FNCAPN(II), II=1, 6) /'E', 'O', 'N', 'T', ' ', '/
```

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
INTRODUCTION

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 the NCAR CGM translator at execution time. For details on how GRAPHCAPs fit into the overall picture of metafile translation, see the "Introduction" of this manual section.

This section is oriented toward the user who is familiar with the operation and instruction set of the target computer graphics display device. Casual users will find that generating a GRAPHCAP from scratch is not an easy task. However, 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 one cannot utilize the GRAPHCAP concept. Instructions for implementation of the translator for devices with a software interface are not included in this manual, but will appear in future editions. The ambitious installer may want to undertake making the necessary changes to the translator code itself in order to implement the translator on a device with a software interface.

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

- INTnnn 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.

**FORMATTING AND ENCODING COORDINATES**

The process of creating coordinates, color/intensities, and other parameters which 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 which 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.

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

4. data_value

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

In most cases you will not know how many input words will be received. So how can you enter enough data values greater than zero to cover the input stream? If you request too many data values, unexpected and unpredictable things will happen. However, you do not need to specify enough entries for all input data. The format table describes one coordinate pair and will cycle until all input data 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:
<table>
<thead>
<tr>
<th>Bit Start</th>
<th>Bit Count</th>
<th>Data Type</th>
<th>Data Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>27</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

More examples are provided in each FORMAT type keyword.

The Encoding Process

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

0 - binary encoding

No change to the formatted data.

1 - ASCII decimal encoding

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

2 - ASCII hexadecimal encoding

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

3 - ASCII octal encoding

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

4 - ASCII TEKTRONIX encoding

This type of encoding is used by the TEKTRONIX 41xx class of devices.

5 - ASCII Real

This encoding translates a bit stream into a printable floating point number. The conversion from the bit stream, which is 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: 100 characters.

DEVICE_TEXT_INIT
Description: Enter text (non-graphics) mode. Sent at termination of the translator.
Type: String.
Default: Null
Maximum number of entries: 100 characters

DEVICE_BATCH
Description: Logical value indicating whether the translator sends the USER_PROMPT and waits for keyboard carriage return. This operation happens at completion of a picture. Batch devices may include laser printers and film devices. Non-batch devices may be graphics display terminals. Batch devices do not prompt for user intervention, they automatically proceed to the next frame.
Type: Logical
Default: FALSE, user intervention is required to advance pictures.
Maximum number of entries: 1

DEVICE_ERASE
Description: String sent to the device when the display surface is to be cleared.
Type: String
Default: Null
Maximum number of entries: 50 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 carriage 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: 50 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 DEVICECOORD_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).

DEVICECOORDLOWERLEFT_X

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

Type: Decimal
Default: Null

Maximum number of entries: 1

DEVICECOORDLOWERLEFT_Y

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

Type: Decimal
Default: Null

Maximum number of entries: 1

DEVICECOORDUPPER_RIGHT_X

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

Type: Decimal
NCAR Graphics Installer's Guide

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" at the beginning of this section. Two examples follow. They describe the required bit positions as P for parity, Xn for an X bit and Yn for a Y bit. The parity bit is not relevant to the encoding process, so those bits are skipped over in these examples. We use the AED512 7-bit binary encoding for the first example.

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

The format table will appear as below:

<table>
<thead>
<tr>
<th>bit_start</th>
<th>bit_count</th>
<th>data_type</th>
<th>data_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>17</td>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>bit_start</th>
<th>bit_count</th>
<th>data_type</th>
<th>data_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>-1</td>
<td>32</td>
</tr>
<tr>
<td>0</td>
<td>16</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>-1</td>
<td>32</td>
</tr>
</tbody>
</table>

Type: Decimal rows with 4 elements per row.
Default: Null
Maximum number of entries: 80 (20 rows * 4 columns)

DEVICE_COORD_ENCODING

Description: The encoding scheme used for device coordinates. See the "FORMATTING and ENCODING" discussion at the beginning of 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

**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" discussion at the start of 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

**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" at the start of 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.
This format would be set up as follows:

<table>
<thead>
<tr>
<th>DEVICE_VECTOR_COUNT_FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>/* bit_start bit_count data_type data_value</td>
</tr>
<tr>
<td>1 7 1 6</td>
</tr>
</tbody>
</table>

Type: Decimal rows with 4 elements per row.
Default: Null
Maximum number of entries: 20 (5 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_INDEX_RANGE_DEFINED keyword must be defined in the GRAPHCAP.
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" at the start of 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" at the start of 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.

August 1987
This format would be set up as follows:

<table>
<thead>
<tr>
<th>DEVICE_COLOR_INDEX_FORMAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>/*</td>
</tr>
<tr>
<td>bit_start bit_count data_type data_value</td>
</tr>
<tr>
<td>1 3 0 3</td>
</tr>
<tr>
<td>4 4 1 3</td>
</tr>
</tbody>
</table>

**Type:** Decimal rows with 4 elements per row.

**Default:** Null

Maximum number of entries: 40 (10 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

**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 "FORMATTING and ENCODING" discussion at the start of 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

**DEVICE_MAP_INTENSITY_FORMAT**

Description: The format used to convert the color/intensity values in the CGM to the format required by the output device. This formatter expects sets of three input data words if in a full color mode and single input data words if in a monochrome mode. Observe the setting of the DEVICE_MAP_MODEL keyword for the mode used by this formatter. See the discussion on "FORMATTING and ENCODING" at the start of 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 bitcount datatype 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

<table>
<thead>
<tr>
<th>bit_start</th>
<th>bit_count</th>
<th>data_type</th>
<th>data_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>-1</td>
<td>32</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>-1</td>
<td>46</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
<td>-1</td>
<td>58</td>
</tr>
</tbody>
</table>

**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:** 30

### DEVICE\_MAP\_INSTRUCTION\_TERMINATOR

**Description:** Terminate the Color Map instruction.  
**Type:** String  
**Default:** Null  
**Maximum number of entries:** 10 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

LINE CONTROL

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

LINE_DRAW_POLY_FLAG

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

This flag indicates the availability, or lack of availability, of a polyline instruction on the device.
When a polyline instruction is available, the LINE_DRAW_INSTRUCTION_START 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:** 10

---

**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:** 5

---

**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:** 10

---

**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:** 5
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: 10

LINE_COLOR_INSTRUCTION_TERMINATOR
Description: Terminate the line color instruction.

Type: String
Default: Null
Maximum number of entries: 10

LINE_WIDTH_INSTRUCTION_START
Description: Change the line width on the display surface.

Type: String
Default: Null
Maximum number of entries: 10

LINE_WIDTH_INSTRUCTION_TERMINATOR
Description: Terminate the line width instruction.

Type: String
Default: Null
Maximum number of entries: 10

LINE_WIDTH_ENCODING
Description: The encoding scheme used for line width values. See the discussion on "FORMATTING and ENCODING" at the start of this section.

Type: Decimal in the range of 0 to 5.
Default: 0 (zero)
NCAR Graphics Installer's Guide

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

**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: 20 (5 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 does not take advantage of hardware markers. Hardware markers were not accurate enough for our test users. 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.

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.

The translator does interpret the following keywords:

MARKER_COLOR_INSTRUCTION_START
Description: Change the marker color.
Type: String
Default: Null
Maximum number of entries: 10

MARKER_COLOR_INSTRUCTION_TERMINATOR
Description: Terminate the color instruction.
Type: String
Default: Null
Maximum number of entries: 10

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

August 1987
These keywords give limited control over device bundle tables. 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 user may modify these bundle tables, but such modification should be documented for the benefit of the users. The tables are coded in an obvious fashion — the first element of each keyword applies to the first table, and so on. For example, the first element in the keyword BUNDLE.LINE_INDEX is the bundle table index used for the bundle that consists of the first elements of each of the keywords BUNDLE.LINE_TYPE, BUNDLE.LINE_WIDTH, and BUNDLE.LINE_COLOR.

**BUNDLE.LINE_INDEX**

Description: Define the indices for the LINE bundle tables.  
Type: Decimal  
Default: 1 2 3 4 5  
Maximum number of entries: 5

**BUNDLE.LINE_TYPE**

Description: Define the line types for the LINE bundle tables.  
Type: Decimal  
Default: 1 2 3 4 5  
Maximum number of entries: 5

**BUNDLE.LINE_WIDTH**

Description: Define the line width scale factors for the LINE bundle tables.  
Type: Floating point  
Default: 1.1.1.1.1.  
Maximum number of entries: 5

**BUNDLE.LINE_COLOR**

Description: Define the line color indices for the LINE bundle tables.  
Type: Decimal  
Default: 1 2 3 4 5  
Maximum number of entries: 5
**BUNDLE_MARKER_INDEX**
Description: Define the indices for the MARKER bundle tables.
Type: Decimal
Default: 1 2 3 4 5
Maximum number of entries: 5

**BUNDLE_MARKER_TYPE**
Description: Define the marker types for the MARKER bundle tables.
Type: Decimal
Default: 1 2 3 4 5
Maximum number of entries: 5

**BUNDLE_MARKER_SIZE**
Description: Define the marker size scale factors for the MARKER bundle tables.
Type: Floating point
Default: 1. 1. 1. 1. 1.
Maximum number of entries: 5

**BUNDLE_MARKER_COLOR**
Description: Define the marker color indices for the MARKER bundle tables.
Type: Decimal
Default: 1 2 3 4 5
Maximum number of entries: 5

**BUNDLE_POLYGON_INDEX**
Description: Define the indices for the FILL AREA bundle tables.
Type: Decimal
Default: 1 2 3 4 5
Maximum number of entries: 5

**BUNDLE_POLYGON_INTERIOR**
Description: Define the fill area interior styles for the FILL AREA bundle tables.
Type: Decimal
Default: 0 0 0 0 0 (hollow)
Maximum number of entries: 5
**BUNDLE_POLYGON_STYLE**
Description: Define the style indices for the FILL AREA bundle tables.
Type: Decimal
Default: 0 0 0 0
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_CEXPAN**
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 which may be filled or hollow. The coordinates are encoded and formatted using the DEVICE_COORD definitions. If there is no polygon instruction on the device, then the polygon color instructions should be the same as the line color instructions.

**POLYGON_INSTRUCTION_START**

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

Type: String

Default: Null

Maximum number of entries: 20

**POLYGON_INSTRUCTION_TERMINATOR**

Description: Terminator for the hardware polygon instruction.

Type: ASCII characters

Default: Null

Maximum number of entries: 10

**POLYGON_COLOR_INSTRUCTION_START**

Description: The instruction to change polygon color.

Type: String

Default: Null

Maximum number of entries: 10
POLYGON_COLOR_INSTRUCTION_TERMINATOR

Description: Terminate the polygon color instruction
Type: String
Default: Null
Maximum number of entries: 10

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_COORDLOWERLEFT_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_COORDLOWERLEFT_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_COORDUPPERRIGHT_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_COORDUPPERRIGHT_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_COORDXOFFSET
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
NCAR Graphics Installer's Guide

**RASTERCOORD_YOFFSET**

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

Type: Decimal
Default: 0 (zero)
Maximum number of entries: 1

**RASTERCOORD_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

**RASTERCOORD_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" discussion at the beginning of 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

\[
\text{RUNCODE} = \text{number of pixels} \times (2^{\text{n}}) + \text{color index}
\]

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" discussion at the beginning of 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.
/*
   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)
NCAR Graphics Installer's Guide

**RASTER_VECTOR_COUNT_ENCODING**

Description: The encoding scheme used for the device raster data. See the "FORMATTING and ENCODING" discussion at the beginning of 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
INTRODUCTION

A FONTCAP is a table describing a character font. This section describes how to generate FONTCAP files. The original ASCII FONTCAP file must be input into the FONTCAP preprocessor to produce a binary version of the FONTCAP, which in turn is used by the NCAR CGM translator at execution time. For details on how FONTCAPs fit into the overall picture of metafile translation, see the "Introduction" at the beginning of this section.

In its original 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 keywords are decimal (integer) values. 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. 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 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.

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.

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_HEIGHT
Description:  The height of the highest character in the collating sequence. This value is given in font units.
Type:        Decimal
Maximum number of entries: 1
CHARACTER_WIDTH
Description: The width of characters in a mono-space font, given in font units. A flag of -1 if this is a variable space font.
Type: Decimal
Maximum number of entries: 1

FONT CLASS

This class of keywords define various character positions in the font.

FONT_CENTER
Description: The horizontal center of the font coordinate system.
Type: Decimal
Maximum number of entries: 1

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.
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_WIDTH**

*Description:* The horizontal distance of the font coordinate system. If the font is variable spaced, then this is the widest character.

*Type:* Decimal

*Maximum number of entries:* 1

**FONT_HEIGHT**

*Description:* The vertical distance covered by the font coordinate system.

*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.

**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 each character in the font. Each character must be in its proper position in the collating sequence and preceded by a CHAR keyword.

A stroke consists of 5 components:

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

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

PEN      The pen up/down bit. Zero indicates pen up (a move); one indicates pen down (draw).

Paint Start bit  The draw at start of vector bit. Interpreted same as PEN.
Only effective for the NCAR DICOMED translator.

Paint End bit  The draw at end of vector bit. Interpreted same as PEN.
Only effective for the NCAR DICOMED translator.

All five components must be included for each stroke. The paint bits may be zero or one at any time unless you intend to use the font for the DICOMED.

Presented below is an example of 2 characters as defined in the FONTCAP file. The X and Y lengths are 5 bits. The PEN and PAINT are 1 bit.

<table>
<thead>
<tr>
<th></th>
<th>x_coord</th>
<th>y_coord</th>
<th>pen</th>
<th>paint_st</th>
<th>paint_ed</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>7</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>33</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>29</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>22</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th></th>
<th>x_coord</th>
<th>y_coord</th>
<th>pen</th>
<th>paint_st</th>
<th>paint_ed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>35</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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 60 strokes allowed per character.
- A maximum of 100 characters may be defined.
- The total stroke count cannot exceed 800.
Required Locally-implemented Subroutines
INTRODUCTION

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

Examples of IMACH, 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 NCAR CGM Translator at each installation. All variable types are given in terms of FORTRAN 77. In the descriptions below, "alloc" represents no errors occurring during the execution of the described routine.

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

ARGGET Get 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.

ARGGET (NUMBER, STRING)

This routine returns the requested command line argument.

INPUT
   NUMBER - INTEGER, argument number requested.

OUTPUT
   STRING - CHARACTER*1 STRING(80), string containing the argument.
BCLRED

**BCLRED (UNIT, IOS, STATUS)**

Close 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 - alloc
  non-zero - error closing the unit

BINRED

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

Transfer "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 - alloc
  non-zero - error reading from the unit

BOPRED

**BOPRED (UNIT, FLNAME, IOS, STATUS)**

Open a file for sequential unformatted reading. The file was created either by the GRAPHCAP or FONTCAP preprocessors 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 - alloc
  non-zero - error opening the unit
FLUS

**FLUS (BUFFER, COUNT, UNIT)**

Flush 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

**FRPRMP (BUFFER, COUNT, IOS, STATUS)**

Send the USER_PROMPT and wait 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.

IARGCT

This function is used only when optional command line processing is installed.

**INTEGER FUNCTION IARGCT (DUMMY)**

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.
MSSG

**MSSG (UNIT, MSGNUM, ADDINF)**

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 in the "Status Message" section, 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

**MTOPEN (UNIT, FLNAME, IOS, STATUS)**

Open 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 - alloc
  - non-zero - error opening the file

READIT

**READIT (BUFFER, MAX)**

Read a character string from standard input and put 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

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

Read 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 - alloc
1 - error in reading
2 - end of file found

WRITIT

WRITIT (STRING, COUNT)

Write 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

**BINCLS (UNIT, IOS, STATUS)**

Close 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 - alloc
  non-zero - error closing the unit

BINOPN

**BINOPN (UNIT, FLNAME, IOS, STATUS)**

Open 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 - alloc
  non-zero - error opening the file

BINWRI

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

Transfer 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 - alloc
  non-zero - error writing
CHRCLS

CHRCLS (UNIT, IOS, STATUS)

Close 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

CHROPN (UNIT, FLNAME, IOS, STATUS)

Open 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

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

Transfer 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

August 1987
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**

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

1. Frame count flag. The **OTHER INFO** is the number of frames processed. No Error.
2. Error opening the metacode file. 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 metacode record. The **OTHER INFO** will be a system dependent error status.
5. Metacode type change. This will occur only if pre-CGM NCAR metacode is mixed with NCAR CGM.
6. An invalid metacode type was detected. The translator attempts to skip and look for another record to use.
7. Invalid metacode 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 a system dependent status value.
16. Error reading the FONTCAP file. The **OTHER INFO** will display a system dependent status value.
17. Error closing the FONTCAP file. The **OTHER INFO** will display a system dependent status value.
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 metacode 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.
Appendix A — Translator ASCII Codes
Appendix A — Translator ASCII Codes

This document describes the ASCII character representations used in the GRAPHCAPs. The list is broken up into two tables. Each character is represented by the code used in GRAPHCAPs, its ASCII name, and its decimal position in the collating sequence.

### Translator ASCII Codes Table 1

<table>
<thead>
<tr>
<th>CHAR</th>
<th>NAME</th>
<th>DEC</th>
<th>CHAR</th>
<th>NAME</th>
<th>DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUL</td>
<td>Null</td>
<td>0</td>
<td>SPC</td>
<td>space</td>
<td>32</td>
</tr>
<tr>
<td>SOH</td>
<td>Start of Heading</td>
<td>1</td>
<td>!</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>STX</td>
<td>Start of Text</td>
<td>2</td>
<td>&quot;</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>ETX</td>
<td>End of Text</td>
<td>3</td>
<td>#</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>EOT</td>
<td>End of Transmission</td>
<td>4</td>
<td>$</td>
<td></td>
<td>36</td>
</tr>
<tr>
<td>ENQ</td>
<td>Enquiry</td>
<td>5</td>
<td>%</td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledgment</td>
<td>6</td>
<td>&amp;</td>
<td></td>
<td>38</td>
</tr>
<tr>
<td>BEL</td>
<td>Bell</td>
<td>7</td>
<td>:</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td>BS</td>
<td>Backspace</td>
<td>8</td>
<td>(</td>
<td></td>
<td>40</td>
</tr>
<tr>
<td>HT</td>
<td>Horizontal Tab</td>
<td>9</td>
<td>)</td>
<td></td>
<td>41</td>
</tr>
<tr>
<td>LF</td>
<td>Line Feed</td>
<td>10</td>
<td>*</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>VT</td>
<td>Vertical Tab</td>
<td>11</td>
<td>+</td>
<td></td>
<td>43</td>
</tr>
<tr>
<td>FF</td>
<td>Form Feed</td>
<td>12</td>
<td>,</td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>CR</td>
<td>Carriage Return</td>
<td>13</td>
<td>-</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>SO</td>
<td>Shift Out</td>
<td>14</td>
<td>.</td>
<td></td>
<td>46</td>
</tr>
<tr>
<td>SI</td>
<td>Shift In</td>
<td>15</td>
<td>/</td>
<td></td>
<td>47</td>
</tr>
<tr>
<td>DLE</td>
<td>Data Link Escape</td>
<td>16</td>
<td>0</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>DC1</td>
<td>Device Control 1</td>
<td>17</td>
<td>1</td>
<td></td>
<td>49</td>
</tr>
<tr>
<td>DC2</td>
<td>Device Control 2</td>
<td>18</td>
<td>2</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>DC3</td>
<td>Device Control 3</td>
<td>19</td>
<td>3</td>
<td></td>
<td>51</td>
</tr>
<tr>
<td>DC4</td>
<td>Device Control 4</td>
<td>20</td>
<td>4</td>
<td></td>
<td>52</td>
</tr>
<tr>
<td>NAK</td>
<td>Negative Acknowledgment</td>
<td>21</td>
<td>5</td>
<td></td>
<td>53</td>
</tr>
<tr>
<td>SYN</td>
<td>Synchronization Character</td>
<td>22</td>
<td>6</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>ETB</td>
<td>End of Transmission Block</td>
<td>23</td>
<td>7</td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>CAN</td>
<td>Cancel</td>
<td>24</td>
<td>8</td>
<td></td>
<td>56</td>
</tr>
<tr>
<td>EM</td>
<td>End of Medium</td>
<td>25</td>
<td>9</td>
<td></td>
<td>57</td>
</tr>
<tr>
<td>SUB</td>
<td>Substitute</td>
<td>26</td>
<td>:</td>
<td></td>
<td>58</td>
</tr>
<tr>
<td>ESC</td>
<td>Escape</td>
<td>27</td>
<td>;</td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>FS</td>
<td>Field Separator</td>
<td>28</td>
<td>&lt;</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>GS</td>
<td>Group Separator</td>
<td>29</td>
<td>=</td>
<td></td>
<td>61</td>
</tr>
<tr>
<td>RS</td>
<td>Record Separator</td>
<td>30</td>
<td>&gt;</td>
<td></td>
<td>62</td>
</tr>
<tr>
<td>US</td>
<td>Unit Separator</td>
<td>31</td>
<td>?</td>
<td></td>
<td>63</td>
</tr>
<tr>
<td>CHAR</td>
<td>NAME</td>
<td>DEC</td>
<td>CHAR</td>
<td>NAME</td>
<td>DEC</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>-----</td>
</tr>
<tr>
<td>@</td>
<td>@</td>
<td>64</td>
<td>'</td>
<td>'</td>
<td>96</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>65</td>
<td>a</td>
<td>a</td>
<td>97</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>66</td>
<td>b</td>
<td>b</td>
<td>98</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>67</td>
<td>c</td>
<td>c</td>
<td>99</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>68</td>
<td>d</td>
<td>d</td>
<td>100</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>69</td>
<td>e</td>
<td>e</td>
<td>101</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>70</td>
<td>f</td>
<td>f</td>
<td>102</td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>71</td>
<td>g</td>
<td>g</td>
<td>103</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>72</td>
<td>h</td>
<td>h</td>
<td>104</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>73</td>
<td>i</td>
<td>i</td>
<td>105</td>
</tr>
<tr>
<td>J</td>
<td>J</td>
<td>74</td>
<td>j</td>
<td>j</td>
<td>106</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>75</td>
<td>k</td>
<td>k</td>
<td>107</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>76</td>
<td>l</td>
<td>l</td>
<td>108</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td>77</td>
<td>m</td>
<td>m</td>
<td>109</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>78</td>
<td>n</td>
<td>n</td>
<td>110</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>79</td>
<td>o</td>
<td>o</td>
<td>111</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>80</td>
<td>p</td>
<td>p</td>
<td>112</td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
<td>81</td>
<td>q</td>
<td>q</td>
<td>113</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>82</td>
<td>r</td>
<td>r</td>
<td>114</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>83</td>
<td>s</td>
<td>s</td>
<td>115</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>84</td>
<td>t</td>
<td>t</td>
<td>116</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>85</td>
<td>u</td>
<td>u</td>
<td>117</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
<td>86</td>
<td>v</td>
<td>v</td>
<td>118</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>87</td>
<td>w</td>
<td>w</td>
<td>119</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>88</td>
<td>x</td>
<td>x</td>
<td>120</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>89</td>
<td>y</td>
<td>y</td>
<td>121</td>
</tr>
<tr>
<td>Z</td>
<td>Z</td>
<td>90</td>
<td>z</td>
<td>z</td>
<td>122</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
<td>91</td>
<td>{</td>
<td>{</td>
<td>123</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
<td>92</td>
<td>}</td>
<td>}</td>
<td>124</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
<td>93</td>
<td>}</td>
<td>}</td>
<td>125</td>
</tr>
<tr>
<td>DEL</td>
<td>delete</td>
<td>95</td>
<td>DEL</td>
<td>delete</td>
<td>127</td>
</tr>
</tbody>
</table>
SECTION V: NCAR Computer Graphics Metafile
Section V: NCAR Computer Graphics Metafile

Introduction ................................................................................................ 117
Record Formatting and NCAR Datatypes.............................................. 117
Supported and Unsupported Elements..................................................... 119
NCAR Computer Graphics Metafile

Introduction

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 uses the Binary Encoding described in the CGM standard. The CGM standard only describes the bit stream that comprises a metafile; it does not describe how the bit stream is 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 old (pre-CGM) NCAR metacode.

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

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

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

The control bits are summarized in their order as follows:

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

Bits 24-32 are currently undefined.

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

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

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

As previously mentioned, the NCAR CGM is a proper subset of those elements mentioned in the CGM standard. The following tables detail which elements are, and which elements are not, supported in the NCAR CGM as of August 1987. "Supported" means that the elements can be generated by the NCAR GKS package, and they are interpreted by the NCAR CGM Translator.
<table>
<thead>
<tr>
<th>Class</th>
<th>Supported Elements</th>
<th>Unsupported Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BEGIN METAFILE</td>
<td>no-op</td>
</tr>
<tr>
<td></td>
<td>END METAFILE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BEGIN PICTURE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BEGIN PICTURE BODY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>END PICTURE</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>METAFILE VERSION</td>
<td>VDC TYPE</td>
</tr>
<tr>
<td></td>
<td>METAFILE DESCRIPTION</td>
<td>INTEGER PRECISION</td>
</tr>
<tr>
<td></td>
<td>METAFILE ELEMENT LIST</td>
<td>REAL PRECISION</td>
</tr>
<tr>
<td></td>
<td>METAFILE DEFAULTS</td>
<td>INDEX PRECISION</td>
</tr>
<tr>
<td></td>
<td>REPLACEMENT</td>
<td>COLOUR PRECISION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COLOUR INDEX PRECISION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAXIMUM COLOUR INDEX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>COLOUR VALUE EXTENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FONT LIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHARACTER SET LIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CHARACTER CODING</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ANNOUNCER</td>
</tr>
<tr>
<td>2</td>
<td>COLOUR SELECTION MODE</td>
<td>SCALING MODE</td>
</tr>
<tr>
<td></td>
<td>VDC EXTENT</td>
<td>LINE WIDTH SPECIFICATION</td>
</tr>
<tr>
<td></td>
<td>BACKGROUND COLOUR</td>
<td>MODE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MARKER SIZE SPECIFICATION</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EDGE WIDTH SPECIFICATION</td>
</tr>
<tr>
<td>3</td>
<td>VDC INTEGER PRECISION</td>
<td>VDC REAL PRECISION</td>
</tr>
<tr>
<td></td>
<td>CLIP RECTANGLE</td>
<td>AUXILIARY COLOUR</td>
</tr>
<tr>
<td></td>
<td>CLIP INDICATOR</td>
<td>TRANSPARENCY</td>
</tr>
<tr>
<td>4</td>
<td>POLYLINE</td>
<td>DISJOINT POLYLINE</td>
</tr>
<tr>
<td></td>
<td>POLYMARKER</td>
<td>RESTRICTED TEXT</td>
</tr>
<tr>
<td></td>
<td>TEXT</td>
<td>APPEND TEXT</td>
</tr>
<tr>
<td></td>
<td>POLYGON</td>
<td>POLYGON SET</td>
</tr>
<tr>
<td></td>
<td>CELL ARRAY</td>
<td>RECTANGLE</td>
</tr>
<tr>
<td></td>
<td>GENERALIZED DRAWING</td>
<td>CIRCLE</td>
</tr>
<tr>
<td></td>
<td>PRIMITIVE</td>
<td>CIRCULAR ARC 3 POINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CIRCULAR ARC 3 POINT CLOSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CIRCULAR ARC CENTRE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CIRCULAR ARC CENTRE CLOSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ELLIPSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ELLIPTICAL ARC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ELLIPTICAL ARC CLOSE</td>
</tr>
<tr>
<td>Class</td>
<td>Supported Elements</td>
<td>Unsupported Elements</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>5</td>
<td>LINE BUNDLE INDEX</td>
<td>CHARACTER SET INDEX</td>
</tr>
<tr>
<td></td>
<td>LINE TYPE</td>
<td>ALTERNATE CHARACTER SET INDEX</td>
</tr>
<tr>
<td></td>
<td>LINE WIDTH</td>
<td>EDGE BUNDLE INDEX</td>
</tr>
<tr>
<td></td>
<td>LINE COLOUR</td>
<td>EDGE TYPE</td>
</tr>
<tr>
<td></td>
<td>MARKER BUNDLE INDEX</td>
<td>EDGE WIDTH</td>
</tr>
<tr>
<td></td>
<td>MARKER TYPE</td>
<td>EDGE COLOUR</td>
</tr>
<tr>
<td></td>
<td>MARKER SIZE</td>
<td>EDGE VISIBILITY</td>
</tr>
<tr>
<td></td>
<td>MARKER COLOUR</td>
<td>PATTERN TABLE</td>
</tr>
<tr>
<td></td>
<td>TEXT BUNDLE INDEX</td>
<td>PATTERN SIZE</td>
</tr>
<tr>
<td></td>
<td>TEXT FONT INDEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TEXT PRECISION</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHARACTER EXPANSION FACTOR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHARACTER SPACING</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TEXT COLOUR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHARACTER HEIGHT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CHARACTER ORIENTATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TEXT PATH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TEXT ALIGNMENT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FILL BUNDLE INDEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INTERIOR STYLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FILL COLOUR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HATCH INDEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PATTERN INDEX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FILL REFERENCE POINT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>COLOUR TABLE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASPECT SOURCE FLAGS</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>ESCAPE</td>
<td>MESSAGE</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>APPLICATION DATA</td>
</tr>
</tbody>
</table>