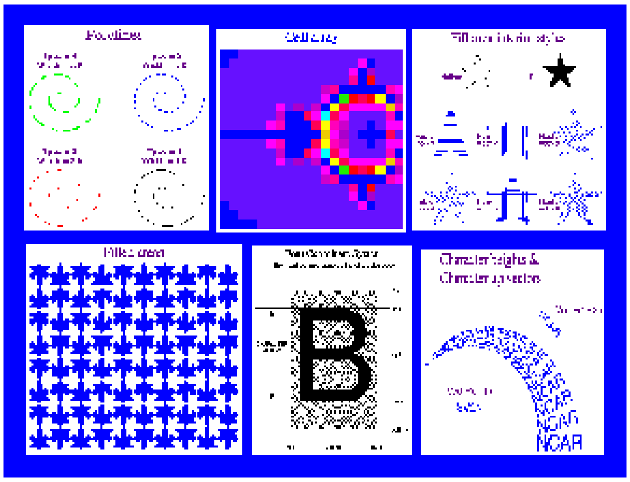
User’s Guide for NCAR GKS-0A Graphics



Contents

Normalization Transformations and Clipping

GKS-0A Output Primitives

Color

X11 Output from NCAR GKS

PostScript Output from NCAR GKS

SPPS Functions

Utility Functions

Introduction to NCAR GKS

What is GKS?

GKS (the Graphical Kernel System) is an ANSI and ISO standard. GKS standardizes two-dimensional graphics functionality at a relatively low level. The primary purposes of the standard are:

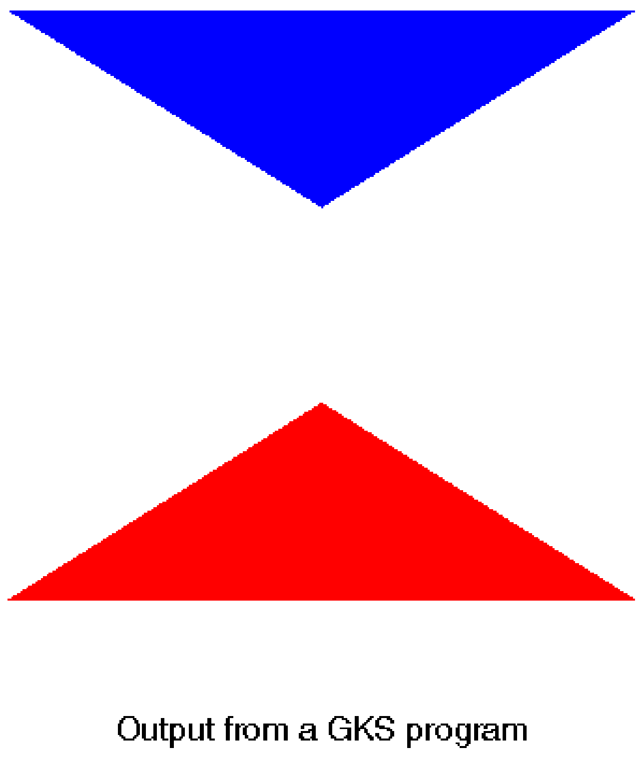
- To provide for portability of graphics application programs.
- To aid in the understanding of graphics methods by application programmers.
- To provide guidelines for manufacturers in describing useful graphics capabilities.

The GKS Standard (as described in the actual standard document: *Computer Graphics - Graphical Kernel System (GKS) Functional Description, ANSI X3.124*-1985) consists of three basic parts:

1. An informal exposition of the contents of the standard which includes such things as how text is positioned, how polygonal areas are to be filled, and so forth.
2. A formalization of the expository material in 1. by way of abstracting the ideas into discrete functional descriptions. These functional descriptions contain such information as descriptions of input and output parameters, precise descriptions of the effect each function should have, references into the expository material in 1., and a description of error conditions. The functional descriptions in this section are language independent.
3. Language bindings. These bindings are an implementation of the abstract functions described in 2. in a specific computer language such as Fortran or Ada or C.

GKS stratifies its functionality into twelve functional levels, depending on the complexity of the graphical input and output. There are four levels of output (m, 0, 1, 2) and three levels of input (A, B, C). NCAR GKS contains a full implementation of the GKS Fortran and C bindings at level 0A.

Figure 1 - A plot generated with a GKS program



NCAR Graphics and GKS

NCAR Graphics is designed to facilitate organizing large amounts of numerical data and producing concise visual representations of those data. With the release of Version 4.0 of NCAR Graphics there are three programming interfaces to the NCAR Graphics package: NCL (NCAR Command Language); Fortran; and C. The programming interfaces provide access to complex graphics utilities like contouring, world map projections, and velocity vectors. For the most part, the C interface is built on top of the Fortran interface.

The complex utilities of the NCAR Graphics package sit on top of GKS-0A. This level of GKS affords more than the minimum level (level m) of output functions, but allows for no graphical input; it cannot, for example, be used to read a point off your screen, but you can output almost any 2D picture to your screen. The NCAR Graphics Low Level Utilities (LLUs) can use the NCAR GKS package that is distributed with the package, or any other GKS package at level 0A or higher. It is possible to use NCAR Graphics without having but a brief knowledge of GKS. However, if you intend to use features of GKS directly, you will need to know more about GKS. In NCAR Graphics version 4.0 and later, direct GKS calls are viewed as LLU calls. For details on mixing LLUs with the HLUs, see the module Mixing HLUs and LLUs. For a discussion of the pros and cons of using LLUs versus the High Level Utilities (HLUs), consult the module HLUs vs. LLUs.

The GKS Documentation Modules

The NCAR GKS documentation modules (of which you are currently reading the "Introduction" module) are all accessible from the " GKS Home Page " module. The NCAR GKS documentation modules contain some site-specific details of NCAR's implementation of GKS-0A. The modules are meant to be used in conjunction with the other on-line documentation for NCAR Graphics. For the purposes of explication, as well as for historical reasons, the NCAR GKS documentation modules focus most heavily on the Fortran binding, although complete details on the C binding are presented.

Described in the NCAR GKS documentation modules are the most commonly used GKS functions of the NCAR Graphics package. GKS at level 0A contains 117 standardized Fortran user entry points and only about half of those are described. If you would like a full understanding of the GKS standard, or more thorough discussions, you should consult the references in Appendix E. All of the references cited are available in the NCAR book libraries.

GKS Output Primitives

GKS at level 0A standardizes a reasonably complete set of functions for displaying 2D images. Users of pre-GKS versions of NCAR Graphics will recognize GKS at level 0A as being a functional extension of what was called the NCAR System Plot Package (NSPP), although with a significantly different user interface. (NSPP contained lower-level graphics calls and was written at NCAR. It did not adhere to any graphics standard and was replaced with GKS in the mid 1980s.) At level 0A, GKS contains functions for drawing lines, markers, filled areas, text, and a function for representing rasterlike images in a device-independent manner. In addition to these basic functions, GKS contains many functions for changing the appearance of the output primitives, such as changing colors, changing line thicknesses, changing marker types and sizes, etc. The functions for changing the appearance of the fundamental drawing functions (output primitives) are called attribute setting functions.

Here is a brief description of the output primitives and their attribute functions. A utility function (that is not part of the GKS Standard) that can be used to save, restore, or set all attribute values in a single call is described in the documentation for NGSRAT. Detailed descriptions of these functions appear in the Output Primitives module.

Polylines

The GKS function for drawing line segments is called polyline. The polyline function takes an array of

X-Y coordinates and draws line segments connecting them. The attributes that control the appearance of a polyline are:

- Linetype, which controls whether the polyline is drawn as a solid, dashed, dotted, or dash-dotted line.
- Linewidth scale factor, which controls how thick the line is.
- Polyline color index, which controls what color the line is.

Polymarkers

The GKS polymarker function allows you to draw marker symbols centered at coordinate points that you specify. The attributes that control the appearance of polymarkers are:

- Marker, which specifies one of five standardized symmetric characters to be used for the marker. The five characters are dot, plus, asterisk, circle, and cross.
- Marker size scale factor, which controls how large each marker is (except for the dot marker).
- Polymarker color index, which specifies what color the marker is.

Text

The GKS text function allows you to draw a text string at a specified coordinate position. The attributes that control the appearance of text are:

- Text font and precision, which specifies what text font should be used for the characters and how precisely their representation should adhere to the settings of the other text attributes.
- Character expansion factor, which controls the height-to-width ratio of each plotted character.
- Character spacing, which specifies how much additional white space should be inserted between characters in a string.
- Text color index, which specifies what color the text string should be.
- Character height, which specifies how large the characters should be.
- Character up vector, which specifies at what angle the text should be drawn.
- Text path, which specifies in what direction the text should be written (right, left, up, or down).
- Text alignment, which specifies vertical and horizontal centering options for the text string.

Fill Area

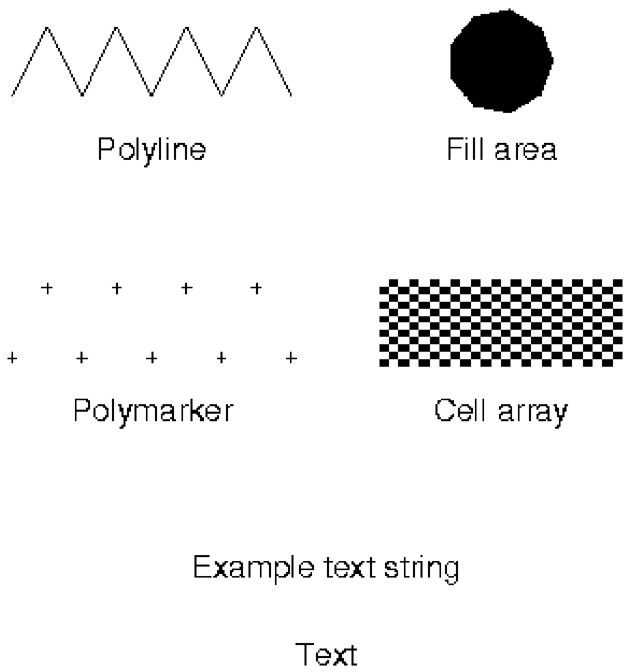
The GKS fill area function allows you to specify a polygonal shape of an area to be filled with various interior styles. The attributes that control the appearance of fill areas are:

- Fill area interior style, which specifies how the polygonal area should be filled: with solid colors or various hatch patterns, or with nothing, that is, a line is drawn to connect the points of the polygon, so you get only a border.
- Fill area style index. If the fill area style is hatch, this index specifies which hatch pattern is to be used: horizontal lines; vertical lines; left slant lines; right slant lines; horizontal and vertical lines; or left slant and right slant lines.
- Fill area color index, which specifies the color of the fill patterns or solid areas.

Cell Array

The GKS cell array function displays rasterlike images in a device-independent manner. The cell array function takes the two corner points of a rectangle that you specify, a number of divisions (M) in the X direction and a number of divisions (N) in the Y direction. It then partitions the rectangle into M x N subrectangles called cells. You assign each cell a color and create the final cell array by coloring each individual cell with its assigned color. At level 0A, cell array has no associated attributes.

Figure 2 - example illustrating the GKS output primitives.

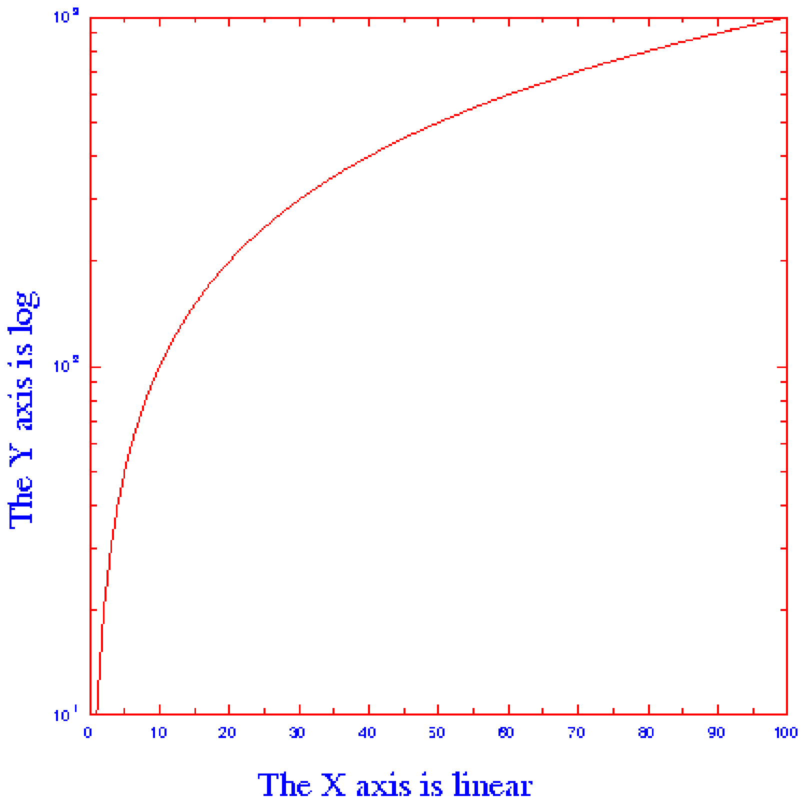


Inquiry Functions

GKS provides a set of inquiry functions that return the current values of GKS variables. Included are variables related to GKS operating states, workstation status, transformations, output attributes, and others. For example, you can inquire about the current value of any GKS output attribute such as the fill area style index, polyline color, polymarker type, and so forth.

Inquiry functions are useful when it is desired to isolate a code so that it can work independently from attribute settings elsewhere. In this case, you can inquire about the value of a variable, save this value, set the value to what the immediate code requires, then reset it to the original value before exiting. In this way, one module will not change the values of variables which are assumed by another module.

Figure 3 - results of code using inquiry functions to isolate drawing text.



GKS Errors

Calls to non-inquiry GKS functions can generate errors. When an error is encountered in such a non-inquiry function, GKS enters an error state, an error handler is called to print the error message, GKS exits the error state, and control is passed back to the calling program. The error handler can be replaced by a user-supplied function, but that is an advanced topic that will not be discussed in the GKS documentation modules. For details on the error handler, consult one of the references in Appendix E.

The inquiry functions of GKS are designed in such a way that the inquiry call will not itself generate an error condition, that is to say GKS never enters an error state and nothing is printed. If the parameter values requested are not available (they do not exist for some reason), an output error indicator is passed back in one of the arguments of the inquiry function. This error parameter indicates whether or not the returned values are valid. If all values are available, the error indicator value is zero. If all state variable values are not available, the error indicator returns an error number which identifies the GKS error condition. These numbers are the same as those used for non-inquiry functions.

An error message list appears in Appendix D.

Links: GKS Index, GKS Home

Coordinate Systems

GKS Coordinate Systems

In addition to its own user coordinate space, NCAR Graphics uses two GKS coordinate systems: world coordinates (WC) and normalized device coordinates (NDC). Both are defined in terms of floating point numbers.

World Coordinates

A world coordinate system is a Cartesian coordinate system that is *linear* along both axes. A world coordinate system can have units of feet, meters, millibars, or whatever engineering units apply to the calculations. World coordinates are expressed as real numbers. You can select a rectangular subspace of world coordinates for output.

Normalized Device Coordinates

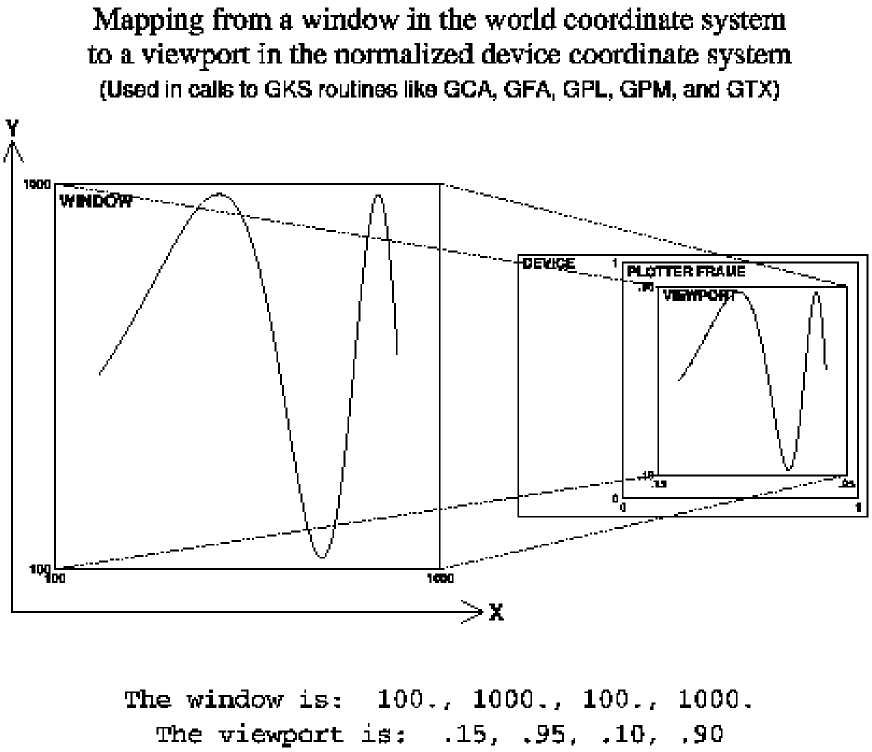
This is a device-independent coordinate system used by GKS as an addressing system for a virtual plotting device; it is a two dimensional Cartesian space that has horizontal and vertical ranges of 0. to 1. inclusive. In GKS, a transformation is performed from a subrectangle of world coordinates (called a "window") to a subrectangle of the normalized device coordinates (NDC). The rectangular subspace of NDC is called a "viewport.". To ensure that squares in world coordinates are mapped onto squares on the output plotting device, the maximum display surface on that device will be the largest square which will fit on the plotting device. The NCAR Graphics CGM interpreter, ctrans, centers the maximum viewport on an output plotting device. X11 output is centered in the output window. PostScript output is centered on the page by default, but options exist for positioning PostScript output anywhere on the page.

In NCAR Graphics, Normalized Device Coordinates are sometimes referred to as "Fractional Coordinates".

Normalization Transformations

The linear transformation between the world coordinate window and the NDC viewport is called a normalization transformation. For example, the horizontal coordinates of the window might be from 100. to 1000. meters. The horizontal coordinates of the viewport might be.15 to.95. Diagram 1 shows a typical GKS normalization transformation.

Diagram 1 - A typical GKS normalization transformation



NCAR Graphics Coordinate Systems

User Coordinates

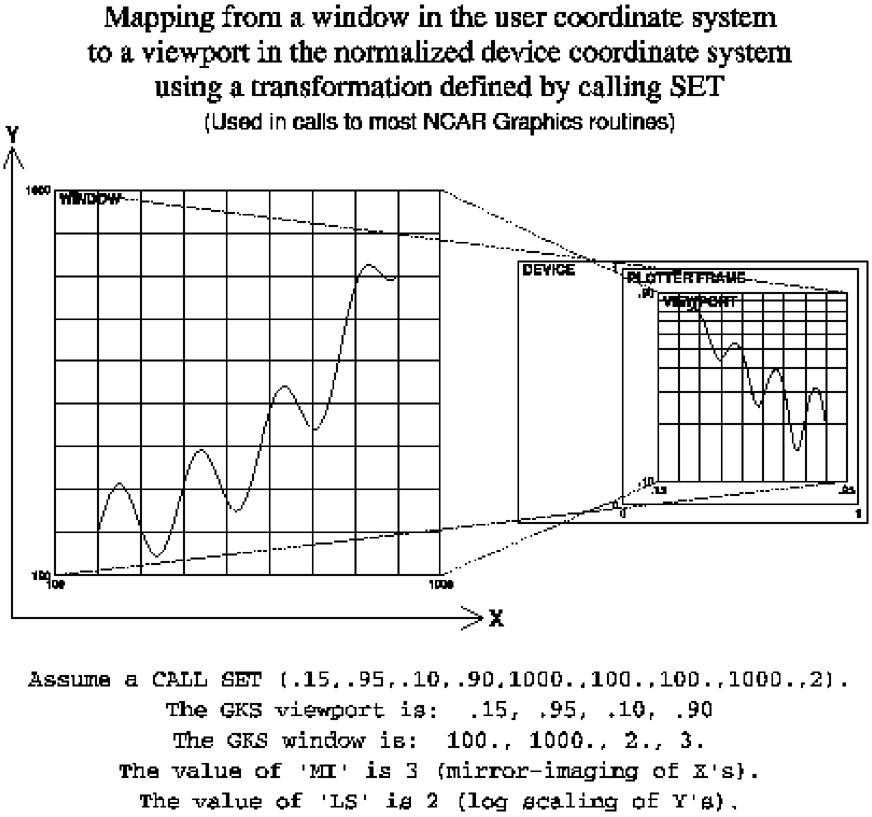
A user coordinate system is any coordinate system in which user calculations are performed. This may be polar coordinates, log scaled coordinates, or any other such coordinate system. If you are working in a user coordinate space that is a Cartesian space with no log scaling or mirror imaging of an axis, then world coordinates and user coordinates are equivalent.

GKS requires that both windows and viewports have linear scaling. GKS also requires that the horizontal coordinate increase from left to right, and that the vertical coordinate increase from bottom to top. Thus, the viewport coordinate (0.,0.) references the lower left corner of a plotting surface. NCAR Graphics provides extensions to GKS world coordinates to allow for mirror imaging of an axis, and log scaling of an axis. If you are using any other coordinate system, then you will have to map it into one of the coordinate systems supported by NCAR Graphics before making GKS calls.

In order to use the log scaling or mirror-imaging features of the NCAR Graphics user coordinate spaces, you must establish your coordinate mapping transformations. The easiest way to do this to by using a SET call. In user coordinates, windows can be defined in which the horizontal coordinate of the window might be meters ranging from 1000. at the left to 100. at the right (mirror imaging), and the vertical coordinate might vary from 10**2 at the bottom to 10**3 at the top in logarithmic scaling.

For a detailed description of the relationship between the GKS coordinate systems and User Coordinates, see NCAR Graphics Fundamentals, UNIX Version, Appendix A.

Diagram 2 - A normalization transformation with mirror imaging and log scaling



Plotter Address Units as Absolute Lengths

In certain places in the documentation for LLUs, reference is made to Plotter Address Units (PAUs). PAUs are integers defined in the range 1 to 1024 and map linearly to normalized device coordinate space. For example, if you wanted a length of .01 in normalized device coordinate space, but were required to specify this as a PAU, then you would use the integer value 102 (.01*1024).

Links: GKS Index, GKS Home

Workstation Functions

Although the NCAR GKS package is a full implementation of the GKS standard at level 0A, only a subset of the functions it contains is commonly used. This documentation module describes the Fortran and C bindings of this subset. It is strictly at the user's discretion which binding to use. In documention the GKS functions, the Fortran binding name will be used in general. In the descriptions of the user entry points in the Fortran and C bindings, the Fortran interface is fully described and a brief synopsis of the C interface is presented. For a full understanding of the GKS standard, or more thorough discussions, consult the references in Appendix E. All of the references cited are available in the NCAR book libraries. The routines described in this section have been grouped according to the order they might appear in a typical GKS application.

Definition of a GKS Workstation

GKS is based on the concept of an abstract graphical workstation. Such a workstation is conceptualized as a rectangular display space of fixed dimension being capable of drawing lines, text, markers, and filled polygons. The connection between the abstract GKS workstations and physical devices is made using the GKS function GOPWK described below.

Workstation Types

Each GKS workstation has a type. Different GKS implementations will support different workstation types. In NCAR GKS, a workstation of type 1 is a private binary encoding conforming to the ANSI Computer Graphics Metafile (CGM) Standard. SCD provides filters, *ncgm2cgm* and *cgm2ncgm*, for converting from the NCAR private encoding of the CGM to and from a generic binary encoding. SCD also supplies a CGM interpreter, *ctrans*, with NCAR Graphics that converts CGM files and plots them on many devices. For information on the filters or *ctrans*, consult the appropriate man page. See Viewing and editing your CGMs and raster images for details on CGM conversion and supported output devices.

In NCAR GKS, a workstation of type 3 is a WISS (Workstation Independent Segment Storage) workstation used for storing picture segments. Workstations of types 7 and 8 are X11 workstations. Workstations of type 10 are ASCII text dumps of graphics instructions. Workstations of types 20-31 are PostScript output workstations.

Workstation Categories

Each GKS workstation type has a unique category that must be one of:

OUTPUT
: Output only

INPUT
: Input only

OUTIN
: Both output and input

WISS
: Workstation independent segment storage (storage for parts of a plot)

MO
: Metafile output

MI
: Metafile input

For NCAR GKS, workstations of type 1 (CGM) are MO workstations. A workstation of type 3 is of category WISS. The only thing you need to know about a WISS workstation (GKS allows only one workstation of category WISS) is that it can be used to store picture segments. WISS is not officially part of GKS at level 0A, but it has been added to NCAR GKS in order to support the Gflash package. For details on Gflash, see NCAR Graphics Fundamentals, UNIX Version and the SCD UserDoc GFLASH - A Graphics Instruction and Manipulation Package. Workstations of type 7 and 8 are of category OUTIN and workstations of types 10 and 20-31 are of category MO.

For more details on GKS workstations, see What you need to know about GKS workstations.

Open and Close GKS

Opening GKS, opening and activating workstations

Before any GKS functions can be called, or before any NCAR Graphics calls can be made, GKS must be opened. Opening GKS initializes all default settings such as character height and marker type. Most GKS functions that are not workstation specific can be called after GKS is open; this includes things like setting line color and thickness, setting fill area interior style and color, setting marker type and size, and so on.

GKS functions that are workstation specific, such as defining color indices, require that the workstation involved be open. In addition, if the GKS output primitives for drawing lines, text, filled areas, markers, and cell arrays are to be sent to a workstation, that workstation must be active.

The call to open GKS is GOPKS. The call to open a workstation is GOPWK. The call to activate a workstation is GACWK.

The following calls are usually the first GKS calls to appear in a GKS program:

```
-------------------------------------------------------
           Argument  |  Type    | Mode   | Dimension
-------------------------------------------------------
CALL GOPKS (ERFILE,  | Integer | Input  |
            BUFA)    | Integer | Input  |
-------------------------------------------------------
CALL GOPWK (WKID,    | Integer | Input  |
            CONID,   | Integer | Input  |
            WKTYPE)  | Integer | Input  |
-------------------------------------------------------
CALL GACWK (WKID)    | Integer | Input  |
-------------------------------------------------------
```

The arguments for the above subroutines are:

ERFILE
: The Fortran unit number to which error messages will be written. Typically this will be unit 6 (standard output).

BUFA
: The dimension of a dynamically managed internal buffer. In NCAR GKS, BUFA is ignored and most people simply specify a zero as its value.

WKID
: A number assigned to a workstation as an identifier that can be used in all subsequent calls to GKS functions that require a workstation identifier. In NCAR GKS, WKID can be any positive integer.

CONID

A connection identifier that has different meanings for different workstation types (see WKTYPE below).

WKTYPE

An identifier specifying the type of output device targeted. Each GKS package has an implementation-dependent set of workstation types that the package supports. The workstation types supported in NCAR GKS follow. In NCAR GKS, workstation types 20-31 refer to PostScript output. The connection ID is irrelevant for the PostScript output workstations. For complete details on the PostScript workstation types, see the PostScript Output module.

1 -- CGM.

The output device will be a metafile whose default name is "gmeta". See the module "Changing a CGM filename" in *NCAR Graphics Fundamentals* to see how to change metafile output names. Also, Appendix A. discusses how to obtain a copy of the example code number 21 that shows how to create two metafiles with different, non-default, names from the same job. There can be at most one open workstation of type 1 at any time. For workstations of type 1, the connection ID is the Fortran logical unit number to be used for the writes to the metafile.

3 -- WISS

This stands for "Workstation Independent Segment Storage" and should be opened (using GOPWK) before storing graphics in segments. See GFLASH - A Graphics Instruction and Manipulation Package. for more details. For workstations of type 3, the connection ID is the Fortran logical unit number to be used for the writes to the metfile. It is permissible to have workstations of types 1 and 3 open simultaneously, but they must have different connection identifiers.

7 -- Pre-existing X11 window

Directs graphics output to a pre-existing X11 window. The X11 window ID should be be specified as connection ID in the GOPWK call.

8 -- New X11 window.

A new X11 window will appear on your screen when this call is executed. The connection ID is irrelevant for workstations of type 8.

10 -- Text dump of graphics output.

Writes a human-readable ASCII dump of the output. Since output is directed to standard out, the connection identifier is irrelevant for workstations of type 10.

20 -- Color PostScript, portrait mode.
21 -- Color Encapsulated PostScript (EPS), portrait mode.
22 -- Color Encapuslated PostScript Interchange (EPSI) format, portrait mode.
23 -- Monochrome PostScript, portrait mode.
24 -- Monochrome Encapsulated PostScript (EPS), portrait mode.
25 -- Monochrome Encapuslated PostScript Interchange (EPSI) format, portrait mode.
26 -- Color PostScript output in landscape mode.
27 -- Color Encapsulated PostScript (EPS), landscape mode.
28 -- Color Encapuslated PostScript Interchange (EPSI) format,
29 -- Monochrome PostScript output in landscape mode.
30 -- Monochrome Encapsulated PostScript (EPS), landscape mode.
31 -- Monochrome Encapuslated PostScript Interchange (EPSI) format, landscape.

Errors:

1, 8, 20, 21, 22, 23, 24, 25, 26, 29, 33, 35

```
--------------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gopen_gks(
               const char  *err_file,  /*   name of error file     */
               size_t      mem_unit    /*   units of memory        */
);

void gopen_ws(
               Gint        ws_id,      /*  workstation identifier  */
               const char  *conn_id,   /*  connection identifier   */
               Gint        ws_type     /*  workstation type        */
);

void gactivate_ws(
               Gint        ws_id       /*  workstation identifier  */
);
--------------------------------------------------------------------------------
```

Closing GKS, deactivating and closing workstations

Three GKS calls are required to gracefully exit GKS. They deactivate the workstations (GDAWK), close the workstations (GCLWK), and close the GKS package (GCLKS).

Argument	Type	Mode	Dimension
CALL GDAWK (WKID)	Integer	Input	
CALL GCLWK (WKID)	Integer	Input	
CALL GCLKS			

The argument for the above subroutines is:

WKID
: The WKID identifier must be the same as that used in the opening sets of calls.

Errors:
: 2, 3, 7, 20, 25, 29, 30, 33, 35

```
---------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gdeactivate_ws(
                Gint  ws_id  /*   workstation identifier  */
);
void gclose_ws(
                Gint  ws_id  /*  workstation identifier   */
);
void gclose_gks(
```

```
                void
);
--------------------------------------------------------------------------
```

Updating a Workstation

For the sake of efficiency, NCAR GKS maintains output buffers for all of its workstations. Unless these buffers are flushed, the current state of the picture may not reflect all graphics instructions that have been issued to a workstation. To ensure that a workstation is current with regard to the GKS buffers one needs to update that workstation. Other buffers are maintained in the higher levels of NCAR Graphics that may need to be flushed if you are making NCAR Graphics calls in addition to GKS calls. For more details on how to insure that all graphics instructions are flushed, see the section "Making sure things are current" in the NCAR Graphics Fundamentals module.

```
------------------------------------------------------
          Argument |   Type  |  Mode  |   Dimension
------------------------------------------------------
CALL GUWK (WKID,   | Integer | Input  |
           REGFL)  | Integer | Input  |
------------------------------------------------------
```

The arguments for the above subroutines are:

WKID
: A number identifying the workstation to be updated. WKID must be the same as that used in some previous call to GOPWK.

REGFL
: A flag to specify if the current picture should be regenerated. The possible values are 0 for "postpone" and 1 for "perform". For the workstation types supported in NCAR Graphics, this flag should always be set to 1.

```
-------------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gupd_ws(
             Gint             ws_id,          /*   workstation identifier   */
             Gupd_regen_flag  upd_regen_flag  /*  update regeneration flag  */
);
-------------------------------------------------------------------------------
```

Clearing a Workstation

In order to clear a workstation plotting surface use:

```
--------------------------------------------------------
            Argument |  Type   |  Mode  |  Dimension
--------------------------------------------------------
CALL GCLRWK (WKID,   | Integer |  Input |
             COFL)   | Integer |  Input |
--------------------------------------------------------
```

The arguments for the above subroutine are:

WKID
: A number identifying a workstation to be cleared. This number must be the same as that used in some previous call to GOPWK.

COFL
: Clear the workstation display surface. Options are:

    0 - Check if the display has been written. If not, do not issue a clear surface command.
    1 - Issue a clear surface command whether the surface has been written to or not.

    For CGM generation, GCLRWK is interpreted as a picture termination. If COFL=0 and no output primitives have been called in the current picture, then a call to GCLRWK is a "do nothing"; otherwise, it generates an END PICTURE as well as a BEGIN PICTURE element (as well as other picture initializing elements). If COFL=1, then a call to GCLRWK generates an END PICTURE and a BEGIN PICTURE.

    For X11 windows, clearing a workstation erases all graphics primitives that appear on the screen in the specified window; for PostScript workstations a clear workstation will terminate the current picture (by inserting a "showpage" operator).

Errors:
: 6, 20, 25, 33, 35

```
--------------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gclear_ws(
               Gint        ws_id,      /*   workstation identifier  */
               Gctrl_flag  ctrl_flag   /*  control flag             */
);
--------------------------------------------------------------------------------
```

Inquiry Functions

Get the Open Status of GKS

GKS maintains an internal status flag to indicate what state it is in with regard to workstation activity. A call to GQOPS can be made to determine this status:

Argument	Type	Mode	Dimension
CALL GQOPS (OPSTA)	Integer	Output	

The argument for the above subroutine is:

OPSTA
Returns GKS, workstation, and segment status as per:

0 - GKS is closed
1 - GKS is open
2 - at least one workstation is open
3 - at least one workstation is active
4 - a segment is open

Default:
none
Errors:
none

```
------------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_op_st(
                Gop_st  *op_st  /*  operating state value  */
);
------------------------------------------------------------------------------
```

Links: GKS Index, GKS Home

Normalization Transformations and Clipping

Normalization Transformations

Window

The subroutine GSWN establishes a window, or rectangular subspace, of world coordinates to be plotted. If you desire log scaling or mirror-imaging of axes, use the SET subroutine.

```
-------------------------------------------------
           Argument | Type     | Mode  | Dimension
-------------------------------------------------
CALL GSWN  (TRNUM,  | Integer  | Input |
            XMIN,   | Real     | Input |
            XMAX,   | Real     | Input |
            YMIN,   | Real     | Input |
            YMAX)   | Real     | Input |
```

```
-------------------------------------------------
```

TRNUM
A normalization transformation number. The number of available transformations is implementation specific. In the case of NCAR GKS, two normalization transformations are provided:

0
Refers to the identity transformation in which both the window and viewport have the range of 0. to 1. in both dimensions. This is the default normalization transformation for GKS. This transformation is standrdized within GKS and it cannot be changed. It will always be the identity transformation and attempts to redefine it will produce an error.

1
A normalization transformation in which the window is defined by XMIN to XMAX and YMIN to YMAX.

XMIN
The left horizontal coordinate of the window.
XMIN < XMAX

XMAX
The right horizontal coordinate of the window.

YMIN
The bottom vertical coordinate of the window.
YMIN < YMAX

YMAX
The top vertical coordinate of the window.

Defaults:
The default normalization transformation is transformation 0, the identity transformation.

Errors:
7, 20, 25, 33, 36, 51, 53

```
------------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_win(
              Gint           tran_num,    /*  transformation number  */
              const Glimit   *win_limits  /*  window limits          */
);

------------------------------------------------------------------------------
```

Viewport

The subroutine GSVP establishes a rectangular subspace of normalized device coordinates. If you desire log scaling or morror-imaging of axes, use the SET subroutine.

```
-------------------------------------------------
           Argument | Type     | Mode   | Dimension
-------------------------------------------------
```

```
CALL GSVP  (TRNUM,   | Integer  | Input |
            XMIN,    | Real     | Input |
            XMAX,    | Real     | Input |
            YMIN,    | Real     | Input |
            YMAX)    | Real     | Input |
-------------------------------------------------
```

TRNUM

A normalization transformation number. The number of available transformations is implementation specific. In the case of NCAR GKS, two normalization transformations are provided:

0

Refers to the identity transformation in which both the window and viewport have the range of 0. to 1. in both dimensions. This is the default normalization transformation for GKS. This transformation is standrdized within GKS and it cannot be changed. It will always be the identity transformation and attempts to redefine it will produce an error.

1

A normalization transformation in which the viewport is defined by XMIN to XMAX and YMIN to YMAX.

XMIN

The left horizontal coordinate of the viewport.

XMAX

The right horizontal coordinate of the viewport. 0.<= XMIN < XMAX <= 1.

YMIN

The bottom vertical coordinate of the viewport.

YMAX

The top vertical coordinate of the viewport. 0. <= YMIN < YMAX<= 1.

Defaults:

The default normalization transformation is transformation 0, the identity transformation.

Errors:

7, 20, 25, 33, 36, 51, 54

```
------------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_vp(
             Gint          tran_num,   /*  transformation number   */
             const Glimit  *vp_limits  /*  viewport limits         */
);
------------------------------------------------------------------------------
```

Transformations

To select a predefined transformation from world coordinates to normalized device coordinates, use the following call:

```
---------------------------------------------------
```

```
              Argument |   Type    | Mode    | Dimension
---------------------------------------------------
CALL GSELNT  (TRNUM)   | Integer | Input |
---------------------------------------------------
```

TRNUM

A normalization transformation number. The number of available transformations is implementation specific. In the case of NCAR GKS, two normalization transformations are provided:

0

Selects the identity transformation in which both the window and viewport have the range of 0. to 1. in both dimensions.

1

Selects a normalization transformation in which the window and viewport have been defined by calls to GSWN and GSVP, or the SPPS utility SET.

Default:

If GSELNT is never called, the default normalization transformation is 0 (the identity transformation).

Errors:

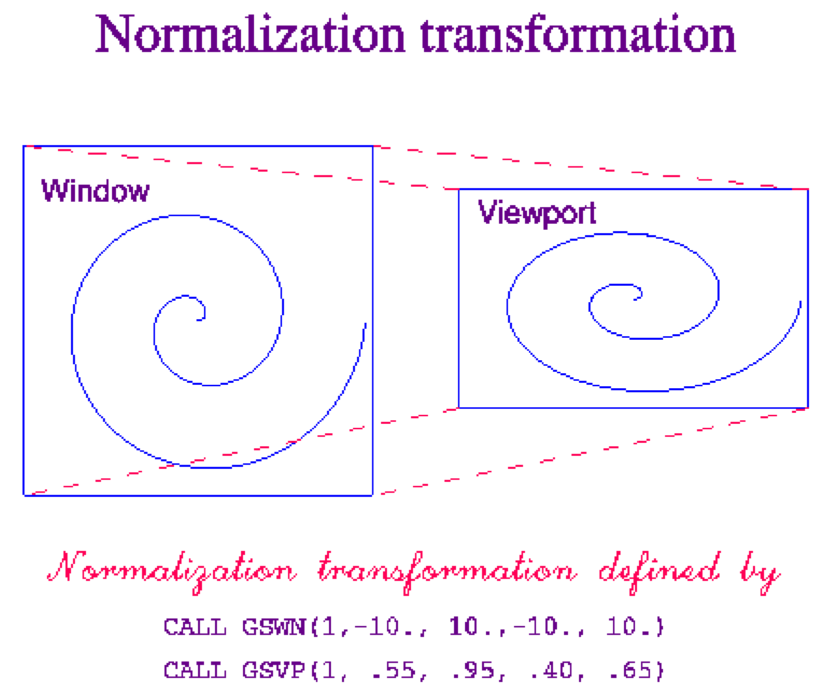
8, 50

```
---------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gsel_norm_tran(
                    Gint  tran_num  /*  transformation number  */
);
---------------------------------------------------------------------------
```

Figure 4 - A GKS normalization transformation



Clipping

The clipping indicator controls whether or not data are displayed outside the boundaries of your defined viewport. If the clipping indicator is off, and you make GKS output calls to plot world coordinate data outside your defined world coordinate window (and your viewport is smaller than the full plotting surface), those data will appear with your plot. If the clipping indicator is on, the data will be clipped to fit your window. To change the clipping indicator, issue the call:

	Argument	Type	Mode	Dimension
CALL GSCLIP	(ICLIP)	Integer	Input	

ICLIP
    An indicator specifying whether clipping is on or off:

0 - Clipping is off. Data outside of the window will be plotted.
1 - Clipping is on. Data outside of the window will not be plotted.

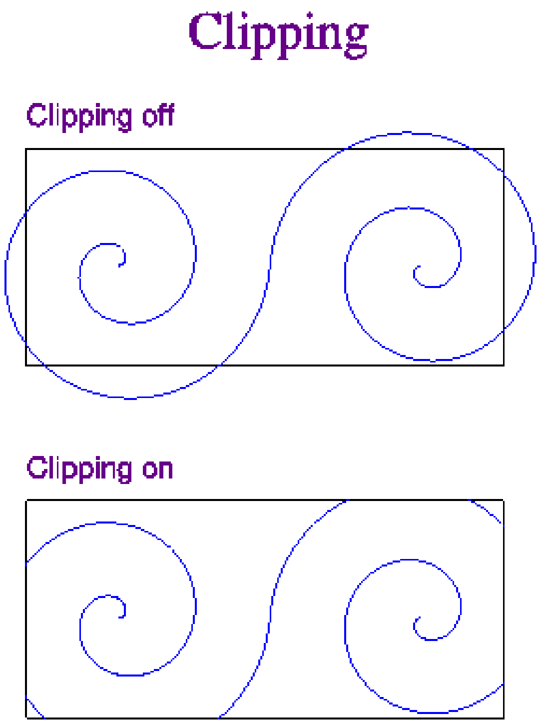
Default:
    By default, clipping is on.
Errors:
    8, 2000

C Synopsis

```
#include <ncarg/gks.h>

void gset_clip_ind(
                   Gclip_ind  clip_ind  /*  clipping indicator  */
);
---------------------------------------------------------------------
```

Figure 5 - Clipping



Inquiry Functions

Get the Number of Supported Normalization Transformations

	Argument	Type	Mode	Dimension
CALL GQMNTN	(ERRIND,	Integer	Output	
	MAXTRN)	Integer	Output	

ERRIND
: Error flag. Gives an integer error number from the Errors list in Appendix D, or a 0 if no error occurred.

MAXTRN
: Gives the maximum number of normalization transformations available in the GKS package. In the case of NCAR GKS, MAXTRN = 2.

Errors:
    8

```
---------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_max_norm_tran_num(
                            Gint  *err_ind,  /*  error indicator  */
                            Gint  *max_num   /*  max trn. number  */
);
---------------------------------------------------------------------
```

Get the Number of the Current Normalization Transformation

	Argument	Type	Mode	Dimension
CALL GQCNTN	(ERRIND,	Integer	Output	
	TRNUM	Integer	Output	

ERRIND
    Error flag. Gives an integer error number from the Errors list in Appendix D, or a 0 if no error occurred.

TRNUM
    Gives the current normalization transformation number. In the case of NCAR GKS the CTNR options are:

    0
        The window and viewport range from 0. to 1. in both the horizontal and vertical dimensions.
    1
        The window and viewport have been set by a previous call to SET or by previous calls to GSWN and GSVP.

Errors:
    8

```
---------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_cur_norm_tran_num(
                        Gint  *err_ind,  /*  error indicator        */
                        Gint  *tran_num  /*  transformation number  */
);
---------------------------------------------------------------------
```

Get the Window and Viewport Associated with a Given Normalization Transformation

If you have defined your normalization transformation by using GSWN and GSVP, then GQNT should be used to retrieve window and viewport limits. If you have used SET to establish your normalization transformation, then you should use GETSET to retrieve window and viewport limits. GETSET will also return the mirror imaging and log scaling information.

```
--------------------------------------------------
           Argument | Type     | Mode    | Dimension
--------------------------------------------------
CALL GQNT  (NTNR,    | Integer  | Input   |
            ERRIND,  | Integer  | Output  |
            WINDOW,  | Real     | Output  | 4
            VIEWPT)  | Real     | Output  | 4
--------------------------------------------------
```

NTNR
: Gives the number of the normalization transformation

ERRIND
: Error flag. Gives an integer error number from the Errors list in Appendix D, or a 0 if no error occurred.

WINDOW
: The horizontal and vertical window limits in the order WXMIN, WXMAX, WYMIN, WYMAX. Window limits are in world coordinates.

VIEWP
: The horizontal and vertical viewport limits in the order VXMIN, VXMAX, VYMIN, VYMAX. Viewport limits are in normalized device coordinates.

Errors:
: 8, 50

```
------------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_norm_tran(
                    Gint   num,        /*  transformation number         */
                    Gint   *err_ind,   /*  error indicator               */
                    Gtran  *norm_tran  /*  normalization transformation  */
);

------------------------------------------------------------------------------
```

Get the Clipping Rectangle and Clipping Indicator

```
--------------------------------------------------
             Argument | Type     | Mode    | Dimension
--------------------------------------------------
CALL GQCLIP  (ERRIND, | Integer  | Output  |
              ICLIP   | Integer  | Output  |
              CLRECT) | Real     | Output  |     4
--------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the Errors list in Appendix D, or a 0 if no error

occurred.

ICLIP

Gives the value of clipping indicator as set by GSCLIP.

0 -- clipping is off, no clipping.
1 -- clipping is on. Data outside of the window will not be plotted.

CLRECT

The four coordinates of the clipping rectangle in the order XMIN, XMAX, YMIN, YMAX. The clipping rectangle is in normalized device coordinates.

Errors:

8,

```
-------------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_clip(
               Gint   *err_ind,        /*  error indicator                    */
               Gclip  *clip_ind_rect   /*  clipping indicator and rectangle   */
);
-------------------------------------------------------------------------------
```

Links: GKS Index, GKS Home

GKS 0A Output Primitives

At the heart of GKS are the *Output Primitives*. These are the GKS functions that actually produce viewable results, like drawing lines, drawing text, and so forth. Each output primitive has an associated set of attributes, such as linewidth and line color for lines. The programming technique is to set values for the output primitives and then call a basic function that will draw the desired primitive using the current settings for the attributes.

GKS difines five output primitives:

1. polyline - for drawing contiguous line segments.
2. polymarker - for marking coordinate positions with various symmetric text symbols.
3. text - for plotting text at various angles and sizes.
4. fill area - for plotting polygonal areas with solid or hatch fill.
5. cell array - for plotting portable raster images.

The documentation modules mentioned above discuss the primitives and the most common functions for setting their attributes. Also discussed are inquiry functions that exist for retrieving current settings of attribute values.

Topics of interest related to the output primitives are:

1. segmentation - saving plot instructions for later use in the same application.
2. aspect source flags - a different way of specifying attribute values.
3. escape functions - functionality unique to NCAR GKS using the GKS "standard way of being non-standard".

Polyline Output

Discussed here is the polyline output primitive *GPL* and the functions that are used to set polyline attributes and retrieve current attribute settings.

Polyline Output Primitive

The polyline output primitive *GPL* causes a series of N-1 straight line segments to be drawn connecting successive points specified by the coordinate pairs (X(I),Y(I)) for I=1,N.

```
------------------------------------------------
           Argument | Type     |  Mode | Dimension
------------------------------------------------
CALL GPL   (N,      | Integer  | Input |
            X,      | Real     | Input |    N
            Y)      | Real     | Input |    N
------------------------------------------------
```

N
: The number of points in the line to be drawn. (N > 1)

X
: An array of dimension N containing the X coordinates of N points to be connected by line segments. These values are in world coordinates.

Y
: An array of dimension N containing the Y coordinates of N points to be connected by line segments. These values are in world coordinates.

Defaults:
: none

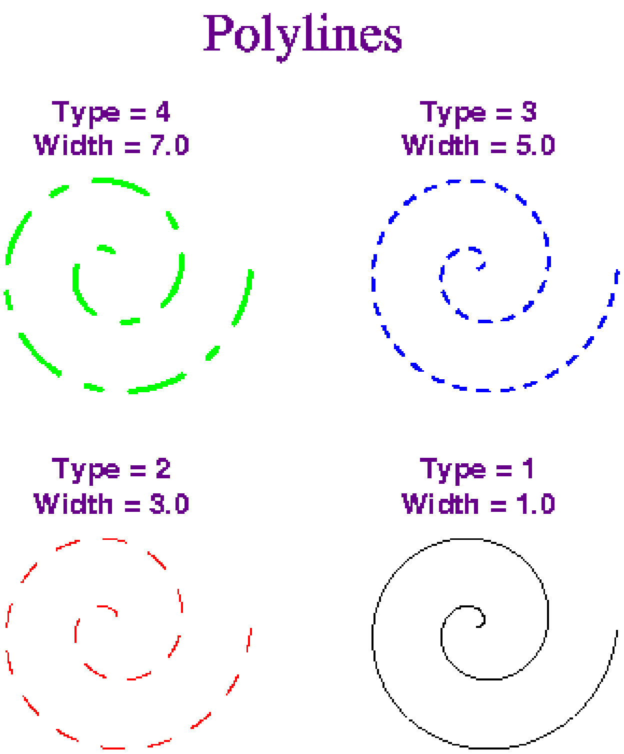
Errors:
: 5, 100

```
-------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gpolyline(
               const Gpoint_list  *point_list  /*  list of points  */
);
-------------------------------------------------------------------------
```

Figure 6 - Examples of polylines



Setting Polyline Output Attributes

Before generating output you can change various polyline attributes from the default values. You can set polyline type and width attributes using GSLN and GSLWSC, respectively.

Set Line Type

```
-------------------------------------------------
           Argument | Type     |  Mode | Dimension
-------------------------------------------------
CALL GSLN  (LTYPE)  | Integer | Input |
-------------------------------------------------
```

LTYPE
    The type of polyline to be drawn. Options are:

< 0 - implementation dependent (not used in NCAR GKS)
1 - solid line
2 - dashed line
3 - dotted line
4 - dashed dotted line
>= 5 - reserved for future standardization (not used in NCAR GKS)

Default:

The default linetype is 1 (solid line).

Errors:

8, 63

```
C Synopsis

#include <ncarg/gks.h>

void gset_linetype(
                   Gint  linetype  /*  linetype  */
);
```

Set Linewidth Scale Factor

Argument	Type	Mode	Dimension
CALL GSLWSC (LWIDTH)	Real	Input	

LWIDTH

A scale factor to control the linewidth of the polyline to be drawn. It must be greater than or equal to 0. LWIDTH is applied to the nominal linewidth on a given device and the linewidths are mapped to the nearest available linewidth representable on that device. See Figure 6 for examples of polylines drawn with different widths. If an output device does not support differing linewidths, then you will see no effect on that device from setting this value.

Default:

The default linewidth scale factor is 1.0.

Errors:

8, 65

```
C Synopsis

#include <ncarg/gks.h>

void gset_linewidth(
                    Gdouble  linewidth  /*  linewidth scale factor  */
);
```

Polyline Inquiry Functions

Get the Line Type

Argument	Type	Mode	Dimension

```
----------------------------------------------------
CALL GQLN  (ERRIND, | Integer |  Output |
            LTYPE)  | Integer |  Output |
----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the Errors list in Appendix D, or a 0 if no error occurred.

LTYPE
: Gives the type of polyline to be drawn as set by GSLN.

Errors:
: 8

```
----------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_linetype(
                  Gint  *err_ind,  /*  error indicator  */
                  Gint  *linetype  /*  linetype         */
);
----------------------------------------------------------------------
```

Get the Linewidth Scale Factor

```
----------------------------------------------------
             Argument | Type     | Mode    | Dimension
----------------------------------------------------
CALL GQLWSC  (ERRIND, | Integer  | Output  |
              LWIDTH) | Real     | Output  |
----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the Errors list in Appendix D, or a 0 if no error occurred.

LWIDTH
: Gives the line width scale factor as set by GSLWSC.

Errors:
: 8

```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_linewidth(
                    Gint     *err_ind,   /*  error indicator         */
                    Gdouble  *linewidth  /*  linewidth scale factor  */
);
--------------------------------------------------------------------------
```

Polymarker Output

Discussed here is the polymarker output primitive *GPM* and the functions that are used to set polymarker attributes and retrieve current attribute settings.

The Polymarker Output Primitive

The polymarker output primitive *GPM* causes N markers to be drawn at the locations specified by the coordinate pairs (X(I),Y(I)) for I=1,N. The markers can be either a dot, a plus sign, an asterisk (the default), a circle, or a cross.

	Argument	Type	Mode	Dimension
CALL GPM	(N,	Integer	Input	
	X,	Real	Input	N
	Y)	Real	Input	N

N
: The number of markers to be drawn. (N > 0)

X
: An array of dimension N containing the X coordinates of the N markers to be drawn, in world coordinates.

Y
: An array of dimension N. The Y coordinates of the N markers to be drawn, in world coordinates.

Defaults:
: none

Errors:
: 5, 100

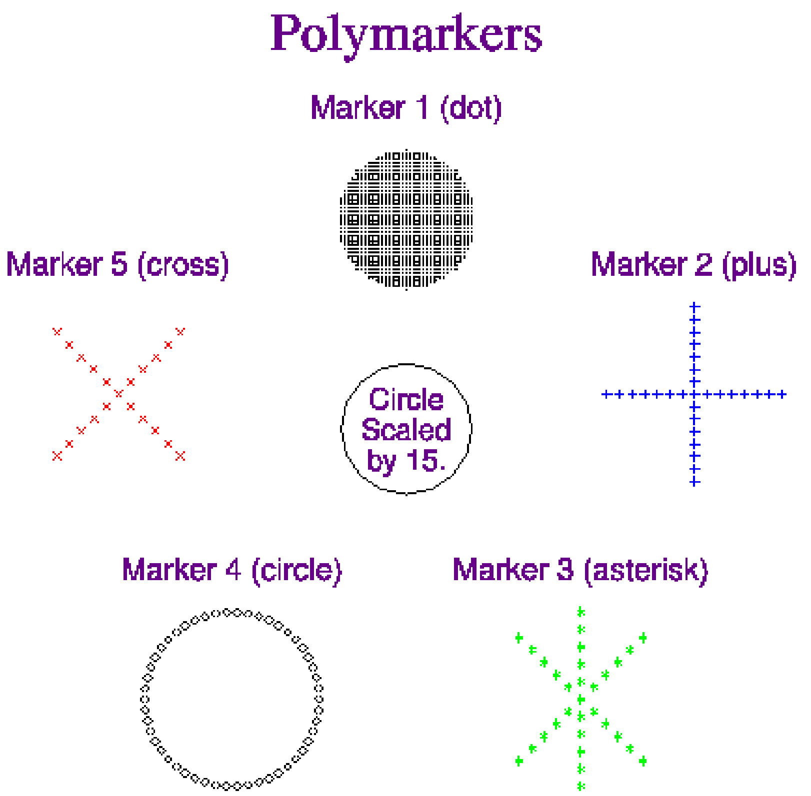
By default, the polymarker type is an asterisk. To select other polymarker types and attributes, see Setting Polymarker Attributes. All standardized polymarkers are symmetric and will be centered on the specified coordinates.

```
------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gpolymarker(
                 const Gpoint_list  *point_list  /*  list of points  */
);
------------------------------------------------------------------------
```

Figure 7 - Examples of polymarkers



Setting Polymarker Output Attributes

You can set polymarker types and sizes using GSMK and GSMKSC, respectively.

Set Polymarker Type

```
-------------------------------------------------
           Argument | Type      |  Mode | Dimension
-------------------------------------------------
CALL GSMK   (MTYPE)  | Integer | Input |
-------------------------------------------------
```

MTYPE

Selects the type of marker to be drawn. Options are:

< 0 - implementation dependent (not used in NCAR GKS)
1 - . (dot)
2 - + (plus)
3 - * (asterisk)
4 - o (circle)
5 - X (cross)
>= 6 - reserved for registration or future standardization

Default:
    The default polymarker is an asterisk.
Errors:
    8, 69

```
---------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_marker_type(
                      Gint  marker_type  /*  marker type  */
);
---------------------------------------------------------------------
```

Set Polymarker Size Scale Factor

```
--------------------------------------------------
             Argument | Type | Mode  |  Dimension
--------------------------------------------------
CALL GSMKSC  (MSZSF)  | Real | Input |
--------------------------------------------------
```

MSZSF
    A scale factor to control the size of the marker. It must be greater than or equal to 0.
Default:
    1.0
Errors:
    8

The GKS standard requires that marker type 1 always be displayed as the smallest possible dot. For this reason a dot marker *will not be scaled*, regardless of the value of MSZSF. To get filled dots of any size and color, use the NGDOTS utility.

```
---------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_marker_size(
                  Gdouble  marker_size  /*  marker size scale factor  */
);
---------------------------------------------------------------------------
```

Polymarker Inquiry Functions

Get the Polymarker Type

```
--------------------------------------------------
           Argument | Type     | Mode    | Dimension
```

```
--------------------------------------------------
CALL GQMK  (ERRIND,  | Integer | Output |
            MTYPE)   | Integer | Output |
--------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in in Appendix D, or a 0 if no error occurred.

MTYPE
: The polymarker type as set by GSMK.

Errors:
: 8

```
-------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_marker_type(
                      Gint  *err_ind,     /*  error indicator       */
                      Gint  *marker_type  /*  current marker type   */
);
-------------------------------------------------------------------------
```

Get the Polymarker Size Scale Factor

```
----------------------------------------------------
             Argument | Type    | Mode    | Dimension
----------------------------------------------------
CALL GQMKSC  (ERRIND, | Integer | Output |
              MSZSF)  | Real    | Output |
----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in in Appendix D, or a 0 if no error occurred.

MSZSF
: Gives the marker size scale factor as set by GSMKSC.

Errors:
: 8

```
----------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_marker_size(
                      Gint  *err_ind,     /*  error indicator            */
                      Gint  *marker_size  /*  marker size scale factor   */
);
----------------------------------------------------------------------------
```

Text Output

Discussed here is the text output primitive *GTX* and the functions that are used to set text attributes and retrieve current attribute settings.

Text Output Primitive

The text output primitive causes a specified text string to be drawn. A number of attribute setting routines are available for selecting the size, font, precision, and orientation of the text.

```
-----------------------------------------------------
          Argument | Type          |  Mode  | Dimension
-----------------------------------------------------
CALL GTX  (X,      | Real          |  Input |
           Y,      | Real          |  Input |
           STRING) | Character     |  Input |
-----------------------------------------------------
```

X
: The X coordinate for the position point for the text string. X is a world coordinate value. See GSTXAL for how a text string is positioned with respect to its position point.

Y
: The Y coordinate for the position point for the text string. Y is a world coordinate value. See GSTXAL for how a text string is positioned with respect to its position point.

STRING
: The string of characters to be drawn.

Defaults:
: none

Errors:
: 5, 101

```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gtext(
           const Gpoint  *text_pos,    /*  text position     */
           const char    *char_string  /*  character string  */
);
-------------------------------------------------------------------------
```

Setting Text Output Attributes

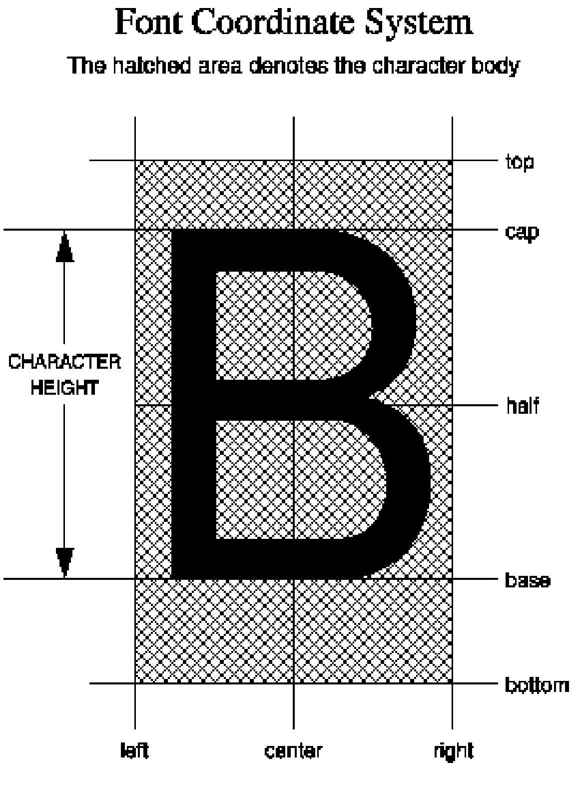
The appearance of text on a workstation is controlled by a number of attributes including:

- Text path (see GSTXP and GQTXP)

- Text alignment (see GSTXAL and GQTXAL)
- Text font and precision (see GSTXFP and GQTXFP)
- Text color index (see GSTXCI and GQTXCI)
- Character height (see GSCHH and GQCHH)
- Character spacing (see GSCHSP and GQCHSP)
- Character up vector (see GSCHUP and GQCHUP)
- Character expansion factor (see GSCHXP and GQCHXP)

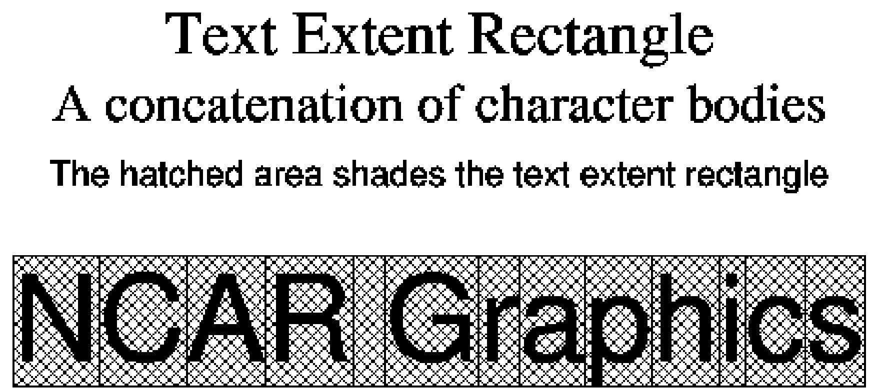
Each character is surrounded by an imaginary box called the character body. The ratio of height to width in a character body is part of the font definition. Most fonts allow the width of the character body to vary from character to character. For example, a capital M will be wider than a lowercase l. Figure 8 shows the character body of a capital B. The character body is delimited by the lines labeled left, right, bottom, and top. The character itself is delimited in the vertical by a base and a cap. The distance between the base and the cap is the character height. The character body usually will contain some white space to the left and right of the character in order to provide pleasing character separations when character bodies are concatenated to produce an output text string.

Figure 8 - Character body



A box that consists of the concatenated character bodies of a series of horizontal or vertical characters defines a text extent rectangle. Text path and alignment attributes relate to the text extent rectangle.

Figure 9 - Text Extent Rectangle



Set Text Path

	Argument	Type	Mode	Dimension
CALL GSTXP	(TXP)	Integer	Input	

TXP

Gives the direction in which a character string is to be drawn. Options are:

0 - draw character n+l to the right of character n
1 - draw character n+1 to the left of character n
2 - draw character n+1 above character n
3 - draw character n+1 below character n

The above, below, left, and right directions are relative to the character up vector (see routine GSCHUP). The text path direction "right" is perpendicular to the up vector direction. To draw a text string at a 45 degree angle, an appropriate character up vector would be (-1.,1.), and the text path would be right.

Normal text writing (horizontal direction moving toward the right) defaults are a character up vector of (0.,1.) and text path of right.

Default:

The default text path is to the right.

Errors:

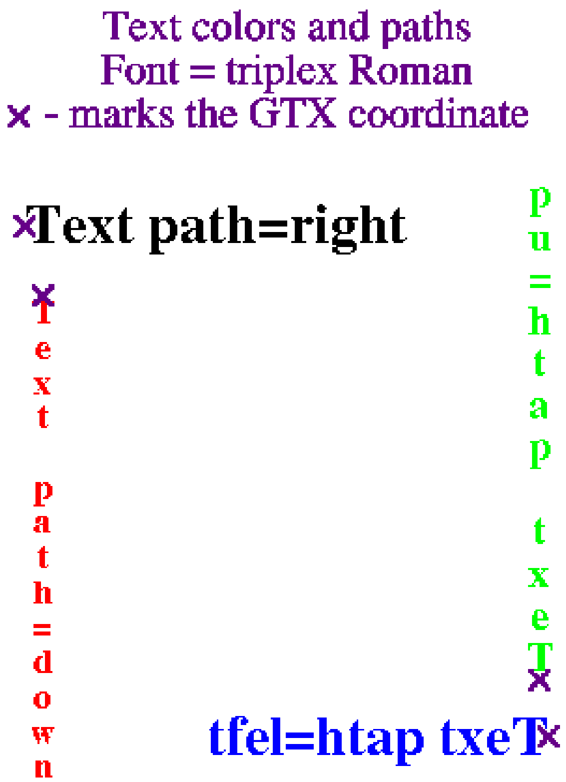
8, 2000

```
----------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_text_path(
                    Gtext_path  text_path  /*  text path  */
);
----------------------------------------------------------------------
```

Figure 10 - text paths and colors



Set Text Alignment

	Argument	Type	Mode	Dimension
CALL GSTXAL	(TXALH,	Integer	Input	
	TXALV)	Integer	Input	

TXALH
: Determines the horizontal alignment of the text to be drawn. Options are:

    0 - normal
    1 - left
    2 - center
    3 - right

TXALV
: Determines the vertical alignment of the text to be drawn. Options are:

    0 - normal
    1 - top
    2 - cap

3 - half
4 - base
5 - bottom

Text alignment is used to indicate a position on the text extent rectangle that will be identified with the positioning point specified in a call to GTX. The angle of rotation at which the text is written is determined by the character up vector (see GSCHUP).

Table 1. Normal Text Alignments Relative to Text Path

Text Path	Normal Horizontal	Normal Vertical
RIGHT	LEFT	BASE
LEFT	RIGHT	BASE
UP	CENTER	BASE
DOWN	CENTER	TOP

Defaults:
The default horizontal alignment is 0; the default vertical alignment is 0.
Errors:
8, 2000

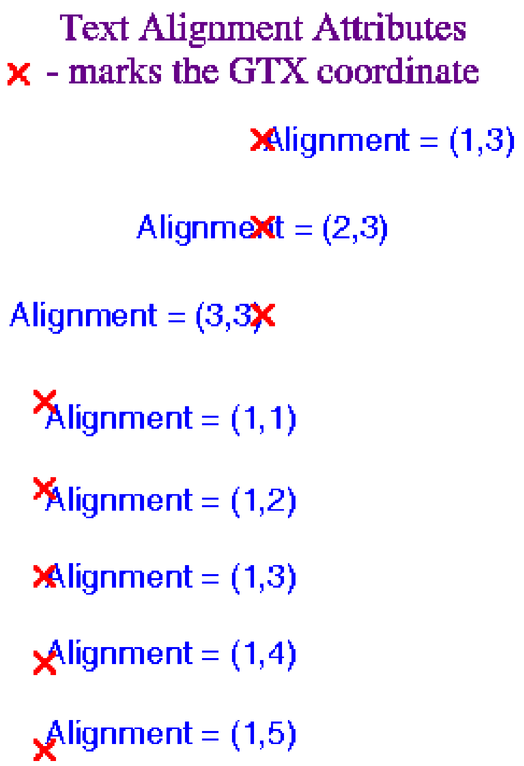
```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_text_align(
                  const Gtext_align  *text_align  /*  text alignment  */
);
--------------------------------------------------------------------------
```

Figure 11 - text alignments

Figure 11, starting from the top and going down, shows the three horizontal positions of left, center, and right combined with a vertical alignment of "half". The next five lines in Figure 11, going down, show the vertical alignments of top, cap, half, base, and bottom for a horizontal alignment of center. Normal text alignment for each text path option is given in Table 1.



Set Text Font and Precision

	Argument	Type	Mode	Dimension
CALL GSTXFP	(FONT,	Integer	Input	
	PREC)	Integer	Input	

FONT

Specifies the text font to be used in subsequent calls to the GTX output primitive. Options are:

1 - Default ASCII font
-2 - Hershey cartographic Roman
-3 - Hershey cartographic Greek
-4 - Hershey simplex Roman
-5 - Hershey simplex Greek
-6 - Hershey simplex script
-7 - Hershey complex Roman
-8 - Hershey complex Greek
-9 - Hershey complex script
-10 - Hershey complex italic
-11 - Hershey complex Cyrillic
-12 - Hershey duplex Roman
-13 - Hershey triplex Roman
-14 - Hershey triplex italic

-15 - Hershey Gothic German
-16 - Hershey Gothic English
-17 - Hershey Gothic Italian
-18 - Hershey math symbols
-19 - Hershey symbol set 1
-20 - Hershey symbol set 2

The Hershey fonts are not standardized by GKS but are locally implemented in NCAR GKS. The Hershey font description files were originally written by A.V. Hershey in the late 1960s and are now made available from the National Institute of Standards and Technology (NIST). All of the Hershey fonts are stroked fonts, that is, the characters are defined by a sequence of line draws. GKS requires that locally implemented fonts be assigned negative font numbers.

When plotting to a PostScript workstation all GKS fonts are mapped onto an equivalent PostScript font if possible.

PREC

Gives an indication as to how faithfully text should be represented when it is plotted with GTX using the font specified by FONT. Options are:

0 - string precision (good)
1 - character precision (better)
2 - stroke precision (best)

Defaults:

FONT=1 and PREC=2 (All text output from NCAR GKS is at stroke precision.)

Errors:

8, 75, 2000

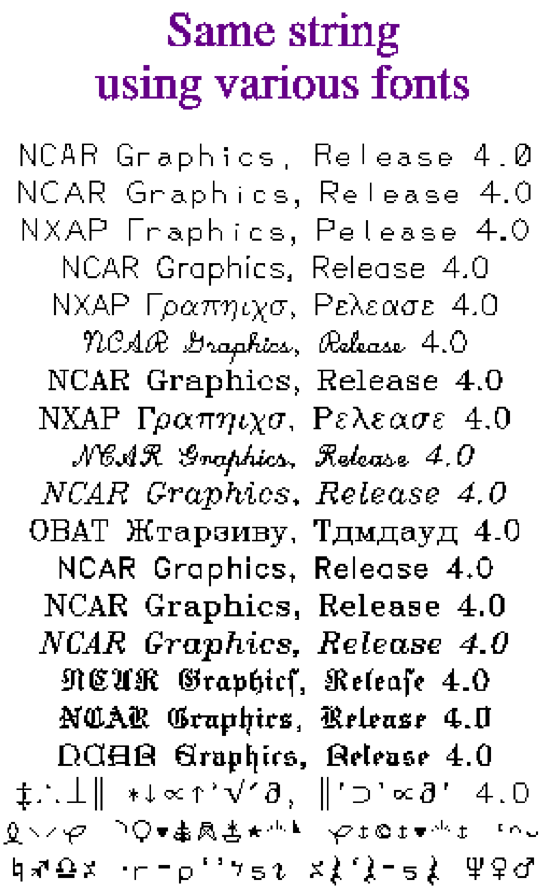
As of release 3.2 of NCAR Graphics, eleven filled fonts were made available via the Plotchar package. PostScript source documentation for Plotchar is available. These fonts are not yet available via GSTXFP.

```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_text_font_prec(
        const Gtext_font_prec  *text_font_prec  /*  text font and prec  */
);
--------------------------------------------------------------------------
```

Figure 12 - same string, different font



Set Character Height

Argument	Type	Mode	Dimension
CALL GSCHH (CHH)	Real	Input	

CHH
: Gives the height in world coordinates at which a character is drawn using the GTX output primitive. CHH must be > 0.

Default:
: The default character height is 0.01

Errors:
: 8, 78

```
C Synopsis

#include <ncarg/gks.h>

void gset_char_ht(
                    Gdouble   char_ht   /*  character height   */
);
```

See Figure 14 for an example of character heights and up vectors.

Set Character Spacing

	Argument	Type	Mode	Dimension
CALL GSCHSP	(CHSP)	Real	Input	

CHSP

Specifies how much additional white space should be inserted between characters drawn using the GTX output primitive. CHSP is specified as a fraction of the character height.

A character spacing of zero concatenates character bodies contiguously. A positive value leaves space between the successive characters. A negative value causes successive characters to overlap.

Default:

The default character spacing is 0.0

Errors:

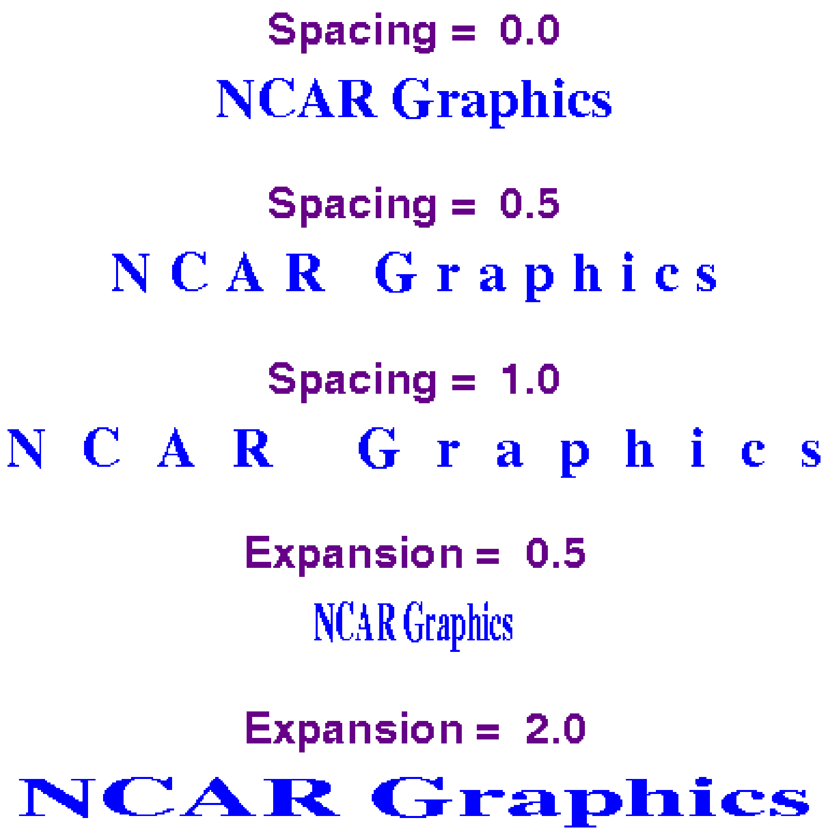
8

```
---------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_char_space(
                     Gdouble  char_space  /*  character spacing  */
);
---------------------------------------------------------------------
```

Figure 13 - Character spacings and expansion factors



Set Character Up Vector

	Argument	Type	Mode	Dimension
CALL GSCHUP	(CHUX,	Real	Input	
	CHUY)	Real	Input	

CHUX
: Gives the X (horizontal) world coordinate of a vector tip.

CHUY
: Gives the Y (vertical) world coordinate of a vector tip. The vector tail is at (0,0). (CHUX, CHUY) cannot be (0,0).

  The coordinate (CHUX, CHUY), relative to (0,0), establishes a vector direction which will be perpendicular to the horizontal centerline of the character string to be drawn. Thus, (CHUX, CHUY) equals (0., 1.) implies normal character positioning in which the character centerline is horizontal and the top of the character is up.

Defaults:
: The default character up vector is (0.,1.)

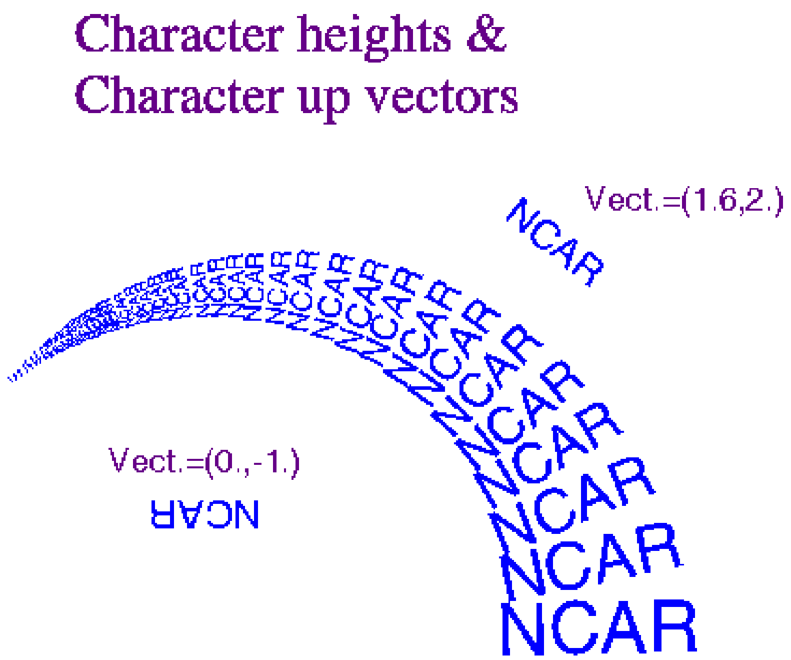
Errors:
: 8, 79

```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_char_up_vec(
                   const Gvec  *char_up_vec  /*  character up vector  */
);
--------------------------------------------------------------------------
```

Figure 14 - Character heights and up vectors



Set Character Expansion Factor

```
-------------------------------------------------
             Argument | Type | Mode    |  Dimension
-------------------------------------------------
CALL GSCHXP   (CHXP)    | Real | Input |
-------------------------------------------------
```

CHXP
: Specifies a deviation of the width to height ratio for characters drawn using the GTX output primitive. CHXP of 10.0 would request a character that is 10 times as wide as normal. CHXP of 0.1 would request characters that are 1/10th as wide as normal. The character height remains unchanged. CHXP must be greater than 0.

Default:
: The default character expansion factor is 1.0 .

Errors:
: 8, 77

```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_char_expan(
                  Gdouble  char_expan  /*  character expansion factor  */
);
--------------------------------------------------------------------------
```

See Figure 13 for examples of character spacings and expansion factors.

Text Inquiry Functions

Get the Text Path

```
---------------------------------------------------
            Argument | Type     | Mode    | Dimension
---------------------------------------------------
CALL GQTXP  (ERRIND, | Integer  | Output  |
             TXP)    | Integer  | Output  |
---------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in Appendix D, or a 0 if no error occurred.

TXP
: Gives the text path as set by GSTXP.

Errors:
: 8

```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_text_path(
                    Gint        *err_ind,   /*  error indicator    */
                    Gtext_path  *text_path  /*  current text path  */
);
--------------------------------------------------------------------------
```

Get the Text Alignment

```
----------------------------------------------------
             Argument | Type     | Mode    | Dimension
----------------------------------------------------
CALL GQTXAL  (ERRIND, | Integer  | Output  |
              TXALH,  | Integer  | Output  |
              TXALV)  | Integer  | Output  |
----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in Appendix D, or a 0 if no error occurred.

TXALH
: Gives the horizontal text alignment as set by GSTXAL.

TXALV
: Gives the vertical text alignment as set by GSTXAL.

Errors:
: 8

```
---------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_text_align(
                     Gint          *err_ind,    /*  error indicator  */
                     Gtext_align  *text_align  /*  text alignment   */
);
---------------------------------------------------------------------------
```

Get the Text Font and Precision

```
-----------------------------------------------------
             Argument  | Type     | Mode     | Dimension
-----------------------------------------------------
CALL GQTXFP  (ERRIND,  | Integer  | Output  |
              FONT,    | Integer  | Output  |
              PREC)    | Integer  | Output  |
-----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in Appendix D, or a 0 if no error occurred.

FONT
: Gives the type of font as set by GSTXFP.

PREC
: Gives the text precision as set by GSTXFP.

Errors:
: 8

```
---------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_text_font_prec(
                   Gint              *err_ind,   /*  error indicator     */
                   Gtext_font_prec  *font_prec  /*  font and precision  */
);
---------------------------------------------------------------------------
```

Get the Character Height

```
---------------------------------------------------
            Argument | Type     | Mode    | Dimension
---------------------------------------------------
CALL GQCHH  (ERRIND, | Integer  | Output  |
             CHH)    | Real     | Output  |
---------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in Appendix D, or a 0 if no error occurred.

CHH
: Gives the character height as set by GSCHH.

Errors:
: 8

```
---------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_char_ht(
                  Gint     *err_ind,  /*  error indicator   */
                  Gdouble  *char_ht   /*  character height  */
);
---------------------------------------------------------------------
```

Get the Character Spacing

```
----------------------------------------------------
             Argument | Type     | Mode    | Dimension
----------------------------------------------------
CALL GQCHSP  (ERRIND, | Integer  | Output  |
              CHSP)   | Real     | Output  |
----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the Errors list in Appendix D, or a 0 if no error occurred.

CHSP
: Gives the character spacing as set by GSCHSP.

Errors:
: 8

```
------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_char_space(
                     Gint     *err_ind,    /*  error indicator    */
                     Gdouble  *char_space  /*  character spacing  */
);
------------------------------------------------------------------------
```

Get the Character Up Vector

```
-----------------------------------------------------
             Argument | Type     | Mode    | Dimension
-----------------------------------------------------
CALL GQCHUP  (ERRIND, | Integer  | Output  |
              CHUX,   | Real     | Output  |
              CHUY)   | Real     | Output  |
-----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in Appendix D, or a 0 if no error occurred.

CHUX
: Gives the X (horizontal) world coordinate of a vector tip.

CHUY
: Gives the Y (vertical) world coordinate of a vector tip. The vector tail is at (0,0).

Errors:
: 8

```
------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_char_up_vec(
                      Gint  *err_ind,     /*  error indicator      */
                      Gvec  *char_up_vec  /*  character up vector  */
);
------------------------------------------------------------------------
```

Get the Character Expansion Factor

```
-----------------------------------------------------
             Argument | Type     | Mode    | Dimension
-----------------------------------------------------
CALL GQCHXP  (ERRIND, | Integer  | Output  |
              CHXP)   | Real     | Output  |
-----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in Appendix D, or a 0 if no error occurred.

CHXP
: Gives the width-to-height ratio of a character as set by GSCHXP.

Errors:
: 8

```
-----------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_char_expan(
                     Gint     *err_ind,    /*  error indicator   */
                     Gdouble  *char_expan  /*  expansion factor  */
```

```
);
-----------------------------------------------------------------------
```

Links: GKS Index, GKS Home

Filled Area Output

Discussed here is the fill area output primitive *GFA* and the functions that are used to set fill area attributes and retrieve current attribute settings.

Fill Area Primitive

The fill area output primitive takes a polygon as input and fills it with a solid color or cross hatching. The GKS standard sets hollow fill as the default. However, due to numerous complaints over many years, the default fill in the NCAR GKS implementation has been set to solid fill.

```
------------------------------------------------
          Argument | Type      |  Mode  | Dimension
------------------------------------------------
CALL GFA   (N,     | Integer   | Input  |
            X,     | Real      | Input  |    N
            Y)     | Real      | Input  |    N
------------------------------------------------
```

N
: The number of points in the polygon to be filled. (N > 2)

X
: An array of dimension N containing the X world coordinates of the polygon.

Y
: An array of dimension N containing the Y world coordinates of the polygon.

The area to be filled is delimited by a series of straight line segments connecting the successive points represented by (X(1), Y(1)), (X(2), Y(2)),..., (X(N), Y(N)). The last point in the polygon is connected to the first point with a straight line segment in the case that coordinate (X(N), Y(N)) is not equal to coordinate (X(l), Y(l)).

Defaults:
: none

Errors:
: 5, 100

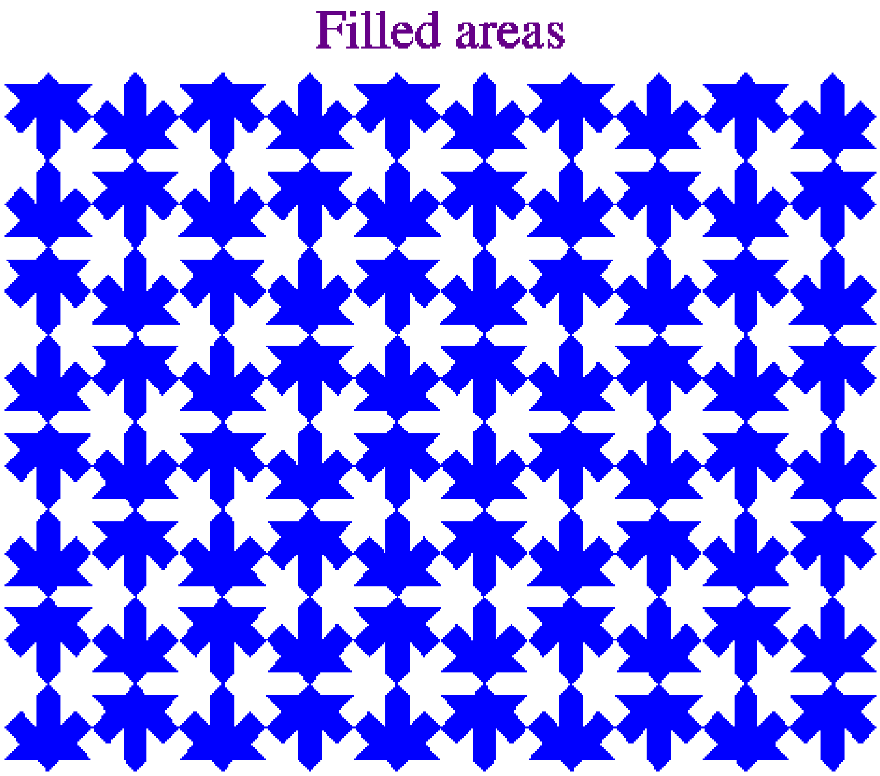
```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gfill_area(
```

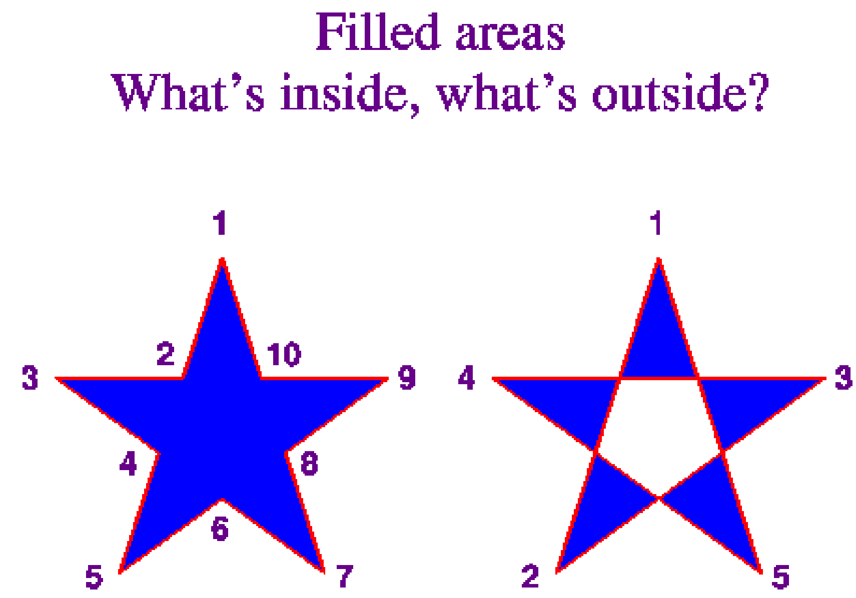
```
                  const Gpoint_list   *point_list  /*  list of points   */
);
-------------------------------------------------------------------------------
```

Figure 15 - Filled areas



Given a polygon whose edges cross over each other, it becomes ambiguous as to what constitutes the "inside" of the polygon. The algorithm used in GKS is as follows: for a given point, create a straight line starting at that point and going to infinity in any direction. If the number of intersections between the straight line and the polygon is odd, the point is within the polygon; otherwise it is outside. If the straight line passes a polygon vertex tangentially, the intersection count is not affected. If a point is within the polygon, it is included in the area to be filled. Figure 16 illustrates the effects of this algorithm. The polygon on the left has ten distinct vertices as labeled, and the inside is as shown. The polygon on the right has five distinct vertices as labeled, and the edges cross one another. The inside is as shown.

Figure 16 - Fill area algorithm



Setting Fill Area Output Attributes

Set Fill Area Interior Style

	Argument	Type	Mode	Dimension
CALL GSFAIS	(INTS)	Integer	Input	

INTS
: Gives the style of fill to be used in subsequent calls to GFA. Options are:

    0 - hollow fill
    1 - solid fill
    2 - pattern fill
    3 - hatch fill

  GKS subroutine GSFASI can *then* be used to set the particular workstation dependent hatch and pattern options which may be provided.

Default:
: The default interior style in NCAR GKS is solid fill. This is in exception to the default specified in the GKS Standard (which is hollow fill).

Errors:
: 8, 2000

```
---------------------------------------------------------------------------
C Synopsis
```

```
#include <ncarg/gks.h>

void gset_fill_int_style(
             Gfill_int_style  fill_int_style  /*  fill interior style  */
);
---------------------------------------------------------------------------
```

Figure 17 below illustrates various fill area interior styles.

Set Fill Area Style Index

```
---------------------------------------------------
             Argument | Type     |  Mode | Dimension
---------------------------------------------------
CALL GSFASI  (STYLI)  | Integer | Input |
---------------------------------------------------
```

STYLI

Allows for the selection of workstation-dependent options for hatch or pattern fills. Pattern fill options are not supported in NCAR Graphics. Hatch fill options are:

1 - horizontal lines
2 - vertical lines
3 - lines of positive slope
4 - lines of negative slope
5 - horizontal and vertical lines
6 - lines of positive and negative slope

Default:

The default hatch fill is horizontal lines.

Errors:

8, 84

```
------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_fill_style_ind(
                         Gint  fill_style_ind  /*  fill style index  */
);
------------------------------------------------------------------------
```

Figure 17 - Fill area interior styles

Fill Area Inquiry Functions

Get the Fill Area Interior Style

```
----------------------------------------------------
              Argument | Type      | Mode     | Dimension
----------------------------------------------------
CALL GQFAIS   (ERRIND, | Integer   | Output   |
               INTS)   | Integer   | Output   |
----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in in Appendix D, or a 0 if no error occurred.

INTS
: Gives the style of fill as set by GSFAIS.

Errors:
: 8

```
---------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>
```

```
void ginq_fill_int_style(
            Gint             *err_ind,        /*  error indicator      */
            Gfill_int_style  *fill_int_style  /*  fill interior style  */
);
---------------------------------------------------------------------------
```

Get the Fill Area Style Index

```
-----------------------------------------------------
             Argument  | Type     | Mode    | Dimension
-----------------------------------------------------
CALL GQFASI  (ERRIND,  | Integer  | Output  |
              STYLI)   | Integer  | Output  |
-----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in in Appendix D, or a 0 if no error occurred.

STYLI
: Gives the fill area style index as set by GSFASI.

Errors:
: 8

```
-------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_fill_style_ind(
                    Gint  *err_ind,        /*  error indicator   */
                    Gint  *fill_style_ind  /*  fill style index  */
);
-------------------------------------------------------------------------
```

Links: GKS Index, GKS Home

Cell Array Output

Discussed here is the cell array output primitive *GCA*. Cell array has no attributes.

Cell Array Primitive

The cell array output primitive starts with a rectangle specified by two diagonally opposite corner points; this initial rectangle is subdivided into subrectangles by specifying an arbitrary number of divisions in the X and Y directions; each subrectangle (or cell) is assigned a color and drawn with that color.

```
-----------------------------------------------------
```

```
          Argument | Type     |  Mode | Dimension
----------------------------------------------------
CALL GCA  (XP1,    | Real     | Input |
           YP1,    | Real     | Input |
           XP2,    | Real     | Input |
           YP2,    | Real     | Input |
           NXCLR,  | Integer  | Input |
           NYCLR,  | Integer  | Input |
           IXCLR,  | Integer  | Input |
           IYCLR,  | Integer  | Input |
           NXDO,   | Integer  | Input |
           NYDO,   | Integer  | Input |
           CLRIA)  | Integer  | Input | NXCLR x NYCLR
-----------------------------------------------------
```

XP1
: X world coordinate of point 1 (see below).

YP1
: Y world coordinate of point 1 (see below).

XP2
: X world coordinate of point 2 (see below).

YP2
: Y world coordinate of point 2 (see below).

NXCLR
: X dimension of the color index array CLRIA.

NYCLR
: Y dimension of the color index array CLRIA.

IXCLR
: The index of the starting column in index array CLRIA.

IYCLR
: The index of the starting row in index array CLRIA.

NXDO
: Number of cell elements to draw in the X dimension.

NYDO
: Number of cell elements to draw in the Y dimension.

CLRIA
: An array of color indices dimensioned NXCLR x NYCLR which is used to determine the color of each cell drawn.

Defaults:
: none

Errors:
: 5, 84

Points 1 and 2 must be diagonally opposite corner points of a rectangle to be divided into NXDO cells in the X dimension and NYDO cells in the Y dimension.

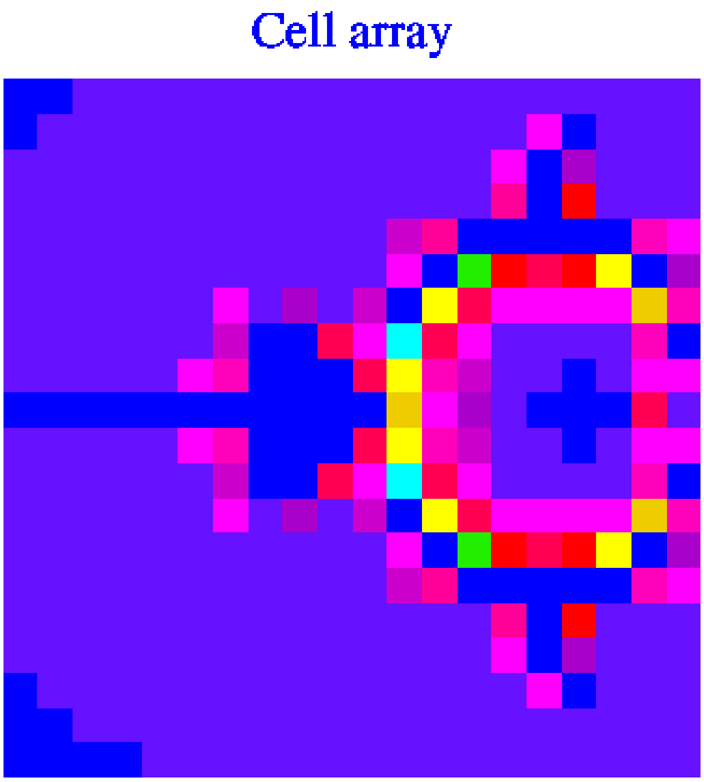
```
-----------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gcell_array(
                 const Grect     *rect,       /*  cell rectangle  */
```

```
                   const Gpat_rep   *colr_array  /*  color array     */
);
------------------------------------------------------------------------
```

Figure 18 - Cell array



Links: GKS Index, GKS Home

Segmentation

In addition to the GKS-0A functions, you may wish to use the segmentation functions provided in GKS at level 2A for storing instructions in segments. You may use those segments in several ways: by inserting them into subsequent pictures; by translating, rotating, and scaling the segments; and by specifying whether a segment is visible or not, or whether it is highlighted or not. The NCAR GKS package has been augmented by implementing a select few GKS level 2A segmentation functions to create, insert, and close segments. We recommend that you use the utility package Gflash to access this functionality. For details on Gflash, see NCAR Graphics Fundamentals, UNIX Version and the SCD UserDoc GFLASH - A Graphics Instruction and Manipulation Package. If you have access to a higher-level GKS package than the NCAR package, then you will probably want to use the segmentation functions of that package directly.

Links: GKS Index, GKS Home

Aspect Source Flags (ASFs)

GKS allows for two methods of specifying attributes for output primitives. One way, which is the way that has been discussed exclusively in this guide, is to specify the attributes individually. Using this method, if you want to change the width of a polyline, then you would call GSLWSC to specify the desired width, and this setting would be in effect until GSLWSC was called again.

The other method of specifying attributes for output primitives is called "bundled". Using this method, the attributes are contained in &quotbundle tables". These tables contain settings for all of the attributes associated with a given output primitive. For example, there can be one or more bundle tables for polylines each of which contains settings for linetype, linewidth scale factor, and polyline color index. Using the bundled attribute setting scheme you set a value for the number of the bundle table that should be used to find the desired setting for the attributes.

Clearly the individual attribute setting scheme and the bundled attribute setting scheme cannot both be in effect simultaneously. GKS supplies a means by which you can specify whether a given attribute should come from its individual setting, or from a bundle table. Associated with each output primitive attribute is an "aspect source flag." The legal values for these flags are "1" for "individual" and "0" for "bundled." Each time GKS draws an output primitive, it first looks at the setting of the aspect source flag for the appropriate attributes, and then either uses the individual setting if the aspect source flag value is "1" or the setting from a specified bundle table if the setting is "0".

All default settings for output primitive attribute aspect source flags are "individual" in NCAR GKS. In higher levels of GKS it is possible for you to define your own bundle tables and select them, but at level 0A, this is not allowed. As implemented, using bundle tables in NCAR GKS is of no value and trying to use them is not advised. The only reason that aspect source flags are discussed here is because the default environment in some GKS packages is "bundled" rather than "individual." In the case that you run NCAR Graphics on top of a non-NCAR GKS package, you will want to check on the default setting for aspect source flags, and if it is "bundled", then you should initialize all the flags to "individual" by using the subroutine GSASF described below. If you do not have access to the necessary GKS documentation, then you can use the subroutine GQASF, described below, to determine the values of the aspect source flags and then, if they are bundled, set them to individual by calling GSASF.

Set Aspect Source Flags

```
---------------------------------------------------
             Argument | Type     |  Mode | Dimension
---------------------------------------------------
CALL GSASF   (LASF)   | Integer | Input |    13
---------------------------------------------------
```

LASF
: List of aspect source flags. A value of 0 means bundled and a value of 1 means individual. The ordering of the flags in LASF is:

1 - linetype ASF
2 - linewidth scale factor ASF
3 - polyline color index ASF
4 - marker type ASF
5 - marker size scale factor ASF
6 - polymarker color index ASF
7 - text font and precision ASF
8 - character expansion factor ASF
9 - character spacing ASF
10 - text color index ASF
11 - fill area interior style ASF
12 - fill area style index ASF
13 - fill area color index ASF

Defaults:
In NCAR GKS all values of LASF are initialized to 1. Other GKS packages may initialize these values to 0.

Errors:
8

```
----------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_asfs(
               const Gasfs  *list_asf  /*  list of asfs  */
);
----------------------------------------------------------------
```

Inquire Aspect Source Flags

```
-------------------------------------------------
            Argument | Type     | Mode    | Dimension
-------------------------------------------------
CALL GQASF  (ERRIND, | Integer  | Output  |
            LASF)    | Integer  | Output  |  13
-------------------------------------------------
```

ERRIND
Error flag. Gives an integer error number from the errors list in in Appendix D, or a 0 if no error occurred.

LASF
List of aspect source flags. A value of 0 means bundled and a value of 1 means individual. The ordering of the flags in LASF is:

1 - linetype ASF
2 - linewidth scale factor ASF
3 - polyline color index ASF
4 - marker type ASF
5 - marker size scale factor ASF

6 - polymarker color index ASF
7 - text font and precision ASF
8 - character expansion factor ASF
9 - character spacing ASF
10 - text color index ASF
11 - fill area interior style ASF
12 - fill area style index ASF
13 - fill area color index ASF

Defaults:

In NCAR GKS all values of LASF are initialized to 1. Other GKS packages may initialize these values to 0.

Errors:

8

```
---------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_asfs(
               Gint   *err_ind,  /*  error indicator      */
               Gasfs  *list_asf  /*  aspect source flags  */
);
---------------------------------------------------------------------
```

Links: GKS Index, GKS Home

Escape Functions

The GKS escape function provides a standard way of implementing nonstandard GKS features. The escape function call is shown below. An example of how to invoke GESC in the NCAR GKS package to create multiple metafiles in a single job is given in example pgkex21. For information on running GKS example codes, see Appendix A.

```
-------------------------------------------------------
           Argument | Type           | Mode   | Dimension
-------------------------------------------------------
CALL GESC  (FCTID,  | Integer        | Input  |
            LIDR,   | Integer        | Input  |
            IDR,    | Character *80  | Input  |    LIDR
            MXODR,  | Integer        | Input  |
            LODR,   | Integer        | Input  |
            ODR)    | Character *80  | Input  |    LODR
-------------------------------------------------------
```

FCTID

A function identifier.

LIDR

Dimension of the IDR input data record array.

IDR

Input data record.

MXODR

Maximum length of the ODR output data record array.

LODR

Dimension of the ODR output data record array.

ODR

Output data record.

Errors:

8, 180

```
---------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gescape(
    Gint                     func_id,      /*  function identifier      */
    const Gescape_in_data   *in_data,      /*  input data record        */
    Gstore                  *store_data,   /*  storage for output data  */
    Gescape_out_data       **out_data      /*  output data record       */
);
---------------------------------------------------------------------------
```

Distributed with NCAR graphics is a set of user entries that will perform the setting of the escape data records and call the GKS escape function. An escape function relevant to NCAR GKS has meaning only in NCAR GKS, and not in any other GKS packages. You should use the entries NGSETI, NGSETR, or NGSETC to make your escape calls. See the man pages for those utilities (which exist only in Version 4.0 or later). See also the man page for *ngmisc_params* which describes all of the available parameters. Here is a summary of the escape functions that exist in NCAR GKS:

Table 1: Summary of escapes in NCAR GKS

#	Description	Notes	ngmisc param.
-1386	Flag whether clipping rectangles in segments are transformed by the segment transformation		IC
-1387	Temporarily close an NCGM.	ncgm only	none
-1388	Set/save/restore primitive attribute settings		none
-1389	Reopen an NCGM for appending.	ncgm only	none
-1390	Returns 'NCAR_GKS0A--VERSION_4.0' in ODR		none
-1391	File name to be used for the next opened file (using GOPWK) of category MO.		ME
-1393	Picture name for the next picture. To name the first picture, call this escape before opening the workstation.	ncgm only	PI
-1394	Sentinal characters for creating segment names.		SE
-1397	Flags whether segments will be deleted when WISS is closed. Default is to delete them.		SS
-1398	Specify maximum number of GKS error messages		ER

	to issue before aborting. Default is ten.		
-1399	Flag to indicate if clipping to the NDC viewport is on. The default is yes (incurring about an 8% overhead.		CL
-1400	Specifies the error allowed in matching the requested colors, expressed as a percentage. Default is 10.	X11 only	PE
-1401	Tells the X driver to specify its own color map rather than sharing with other windows. This is identical to specifying 1 for the color model with escape -1402.	Obsolete	PR
-1402	Tells the X driver which color model to use. First integer is the workstation id. If -1 is used, the escape will apply to the next workstation created. The second integer specifies the color model to use: 0 = shared X colormap 1 = private X colormap 2 = mixed model-use shared as long as possible	X11 only	SC,MC, PC
-1512	Specify spacing between software fill lines.	PS only	FI
-1513	Specify spacing between hatch pattern lines.	PS only	HA
-1514	Maximum size of the operand stack.	PS only	ST
-1515	Maximum number of points in a path.	PS only	PA
-1516	Scale factor for nominal linewidth.	PS only	NO
-1517	Specifies whether background color will fill the entire page or just the viewport. Default is viewport.	PS only	FU
-1518	Specify type of line join. Default is "round".	PS only	JO
-1519	Specify type of line cap. Default is "round".	PS only	CA
-1520	Specify miter limit of line joins. Default Default is 10.	PS only	MI
-1521	Coordinate points for positioning output. Default is largest square on output device.	PS only	LX,LY, UX,UY
-1522	Scale factor for representing coordinates.	PS only	CO
-1523	Flag for selecting RGB or CMYK color model.	PS only	CO

for details on escape functions that apply specifically to X output workstations, see X11 escapes.

For details on escape functions that apply specifically to PostScript output workstations, see PostScript escapes.

If you invoke an escape function that applies to a specific workstation type, it will have no affect on any of the other workstation types.

Links: GKS Index, GKS Home

Color

For all GKS output primitives, color is assigned by an indexing scheme. The indices typically run from 0 to N, where 0 is the background color index and 1 is the foreground color index. Indices are assigned by calls to the GKS routine GSCR. In GSCR, an index is assigned a color value based upon the setting of a red, green, blue (RGB) triplet according to the RGB color model. The indices, not RGB settings, are thereafter used to assign color to the output primitives.

For more details on the use of color in GKS, refer to Color tables and color mapping systems. in the NCAR Graphics Fundamentals documentation module.

Color Setting Functions

Set Color Representation

```
------------------------------------------------------------
           Argument | Type     | Mode  | Dimension
------------------------------------------------------------
CALL GSCR  (WKID,   | Integer  | Input |
            CI,     | Integer  | Input |
            CR,     | Real     | Input |
            CG,     | Real     | Input |
            CB)     | Real     | Input |
------------------------------------------------------------
```

WKID
    A workstation identifier.
CI
    A color index to be assigned a color of CR, CG, and CB.
CR
    An intensity of red between a minimum of 0. and a maximum of 1. inclusive.
CG
    A green intensity between 0. and 1.
CB
    A blue intensity between 0. and 1.
Defaults:
    The default color values are specific to the output device.
Errors:
    7, 20, 25, 33, 35, 36, 92, 96

```
------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_colr_rep(
             Gint              ws_id,      /*  workstation identifier  */
             Gint              colr_ind,   /*  color index             */
             const Gcolr_rep   *colr_rep   /*  color representation    */
);
------------------------------------------------------------------------
```

The color index is an identification tag for an RGB (red, green, blue) color setting. When generating an output primitive, you may specify the color for that primitive by referring to the color index. We recommend that you define all color indices required in a given picture before issuing any output primitives in that picture. If you will be using a fixed set of colors throughout an application, then those color indices are best defined after you activate the workstation. It is important to stress that color index 0 defines the background color. If any color indices are defined, then you should define the background color index 0 -- otherwise you run the risk of having a user-defined color match the default background color.

Set Polyline Color Index

```
--------------------------------------------------
             Argument | Type    | Mode  | Dimension
--------------------------------------------------
CALL GSPLCI  (COLI)   | Integer | Input |
--------------------------------------------------
```

COLI
: An index assigned an RGB (red, green, blue) color in a previous call to the GSCR routine or a workstation dependent default. Allowed indices are non-negative integers. Index 0 is the background color. Index 1 is the foreground color.

All lines drawn with calls to the GPL output primitive will be drawn with the color described by index COLI until GSPLCI is called again and a new index is assigned.

Default:
: The default polyline color index is 1 (foreground color).

Errors:
: 8, 92

```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_line_colr_ind(
                        Gint  line_colr_ind  /*  polyline color index  */
);
--------------------------------------------------------------------------
```

Set Polymarker Color Index

```
---------------------------------------------------
             Argument | Type    | Mode  |  Dimension
---------------------------------------------------
CALL GSPMCI  (COLI)   | Integer | Input |
---------------------------------------------------
```

COLI
: An index assigned an RGB (red, green, blue) color in a previous call to the GSCR routine, or a workstation-dependent default. Allowed indices are non-negative integers. Index 0 is the background color. Index 1 is the foreground color.

All markers drawn with calls to the GPM output primitive will be drawn with the color described by index COLI until GSPMCI is called again and a new index is assigned.

Default:
: The default polymarker color index is 1 (foreground color).

Errors:
: 8, 92

```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_marker_colr_ind(
                   Gint  marker_colr_ind  /*  polymarker color index  */
);
--------------------------------------------------------------------------
```

Set Text Color Index

	Argument	Type	Mode	Dimension
CALL GSTXCI	(COLI)	Integer	Input	

COLI
: An index assigned an RGB (red, green, blue) color in a previous call to the GSCR routine, or a workstation-dependent default. Allowed indices are non-negative integers. Index 0 is the background color. Index 1 is the foreground color.

All text drawn with calls to the GTX output primitive will be drawn with the color described by index COLI until GSTXCI is called again and a new index is assigned.

Default:
: The default text color index is 1 (foreground color).

Errors:
: 8, 92

```
----------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void gset_text_colr_ind(
                        Gint  text_colr_ind  /*  text color index  */
);
----------------------------------------------------------------------------
```

Set Fill Area Color Index

	Argument	Type	Mode	Dimension

CALL GSFACI	(COLI)	Integer	Input

COLI
: An index assigned an RGB (red, green, blue) color in a previous call to the GSCR routine, or a workstation dependent default. Allowed indices are non-negative integers. Index 0 is the background color. Index 1 is the foreground color.

All filled areas drawn with calls to the GFA output primitive will be drawn with the color described by index COLI until GSFACI is called again and a new index is assigned.

Default:
: The default fill area color index is 1 (foreground color).

Errors:
: 8, 92

```
C Synopsis

#include <ncarg/gks.h>

void gset_fill_colr_ind(
                        Gint   fill_colr_ind   /*  fill  color index  */
);
```

Color Inquiry Functions

Get the Color Representation

	Argument	Type	Mode	Dimension
CALL GQCR	(WKID,	Integer	Input	
	CI,	Integer	Input	
	TYPE,	Integer	Input	
	ERRIND,	Integer	Output	
	CR,	Real	Output	
	CG,	Real	Output	
	CB)	Real	Output	

WKID
: Gives the workstation identifier.

CI
: A color index.

TYPE
: A flag to indicate whether the returned color values are those that have been requested or those actually used on the workstation.

0 - Returns the colors set for index CI in a GSCR call, or the default case.
1 - Returns the colors actually used by the workstation to represent the requested color.

ERRIND
: Error flag. Gives an integer error number from the errors list in in Appendix D, or a 0 if no error occurred.

CR
: Gives an intensity of red between a minimum of 0. and a maximum of 1. inclusive.

CG
: Gives a green intensity between 0. and 1.

CB
: Gives a blue intensity between 0. and 1.

Errors:
: 7, 20, 25, 93, 2000

```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_colr_rep(
                   Gint       ws_id,     /*  workstation identifier   */
                   Gint       colr_ind,  /*  color index              */
                   Ginq_type  type,      /*  type of returned values  */
                   Gint       *err_ind,  /*  error indicator          */
                   Gcolr_rep  *colr_rep  /*  color representation     */
);
--------------------------------------------------------------------------
```

Get the Polyline Color Index

```
----------------------------------------------------
             Argument | Type     | Mode     | Dimension
---------------------------------------------------
CALL GQPLCI  (ERRIND, | Integer | Output |
              COLI)   | Integer | Output |
----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in in Appendix D, or a 0 if no error occurred.

COLI
: Gives the polyline color index as set by GSPLCI.

Errors:
: 8

```
---------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_line_colr_ind(
                        Gint  *err_ind,       /*  error indicator       */
                        Gint  *line_colr_ind  /*  polyline color index  */
```

```
);
--------------------------------------------------------------------------
```

Get the Polymarker Color Index

```
---------------------------------------------------
             Argument | Type     | Mode    | Dimension
--------------------------------------------------
CALL GQPMCI  (ERRIND, | Integer | Output |
              COLI)   | Integer | Output |
---------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in in Appendix D, or a 0 if no error occurred.

COLI
: Gives the polymarker color index as set by GSPMCI.

Errors:
: 8

```
--------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_marker_colr_ind(
                   Gint  *err_ind,         /*  error indicator         */
                   Gint  *marker_colr_ind  /*  polymarker color index  */
);
--------------------------------------------------------------------------
```

Get the Text Color Index

```
---------------------------------------------------
             Argument | Type     | Mode    | Dimension
--------------------------------------------------
CALL GQTXCI  (ERRIND, | Integer | Output |
              COLI)   | Integer | Output |
---------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in in Appendix D, or a 0 if no error occurred.

COLI
: Gives the text color index as set by GSTXCI.

Errors:
: 8

```
------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_text_colr_ind(
```

```
                        Gint  *err_ind,       /*  error indicator   */
                        Gint  *text_colr_ind  /*  text color index  */
);
--------------------------------------------------------------------------
```

Get the Fill Area Color Index

```
----------------------------------------------------
             Argument | Type     | Mode    | Dimension
---------------------------------------------------
CALL GQFACI  (ERRIND, | Integer | Output |
              COLI)   | Integer | Output |
----------------------------------------------------
```

ERRIND
: Error flag. Gives an integer error number from the errors list in in Appendix D, or a 0 if no error occurred.

COLI
: Gives the fill area color index as set by GSFACI.

Errors:
: 8

```
----------------------------------------------------------------------------
C Synopsis

#include <ncarg/gks.h>

void ginq_fill_colr_ind(
                      Gint  *err_ind,       /*  error indicator        */
                      Gint  *fill_colr_ind  /*  fill area color index  */
);
----------------------------------------------------------------------------
```

Links: GKS Index, GKS Home

X Output

What is X?

Interactive Display

X is the windowing system used by most UNIX computers. X11 is the latest version of this windowing system. X was designed and developed by MIT (Massachusetts Institute of Technology) in the mid to late 1980s.

X is made up of three distinct parts. First, there is a library of graphical input and output routines for programming X *clients*. An X client is an application that draws graphics to a screen in the X environment, and gets input from the devices attached to that screen. *(The keyboard and mouse in most environments.)* Second, there is an X *server* that runs on the host machine that implements the requests made by an X client using the X library calls. And finally, there is the X protocol, which is the communication protocol used by the library and the server to communicate. Applications using the X driver in NCAR GKS are therefore X client applications.

Producing X Output from NCAR GKS

GKS Workstation Types for X output

X output can be produced from the NCAR GKS package into a pre-existing X window, or NCAR GKS can create the X window for you. The normal mode of operation is to let NCAR GKS create the X window for you, but it is sometimes useful for a user to provide a window for NCAR GKS to draw into.

These two output options are implemented in NCAR GKS as two different workstation types. Workstation type 7 is used to direct graphics output to a pre-existing X window, while workstation type 8 is used to have NCAR GKS create the X window. For an overview of workstation types in NCAR GKS, see the Workstation Functions documentation module.

GKS ESCAPE functions controlling X Output

In what follows, details are given on the escape functions in NCAR GKS that apply specifically to X workstations. These escape functions control aspects of the X output that are device specific and lie outside the scope of the GKS Standard. The user interface to these escape functions is via the entries NGSETI, NGSETR, and NGSETC.

X Color Model

In the default X environment, all windows share a common color table. Many systems can only support a table of 256 entries. This means that, in general, it is not possible to display 256 distinct colors in a single window. If another application is using a bunch of colors, then it may be that the window you are using to display NCAR Graphics will have only a limited number of colors available to it. The escape number -1402 is used to tell NCAR GKS which X Color Model to use to deal with this problem. There are currently three Color Models used in NCAR GKS:

- Shared X Color Model

  If the X Workstation is directed to use the Shared X Color Model, then that X window will use the default color table shared by all applications. If you request more colors, via calls to GSCR, than there are available to the given window, then a color matching algorithm is employed. The idea of the algorithm is that the color in the current color table that is *closest* to the requested color will be selected. *Closest* is defined in terms of the normal distance metric on the RGB cube. If the closest color is farther away than the percentage error allowed times the length of the diagonal of the RGB color cube, then error number -220 is issued together with a warning. The closest color is still assigned for the requested color. The Shared X Color Model is the default for workstations of type 7.

For example, if you want an X workstation to use the shared color table, you would make the following call:

```
CALL NGSETI('SC',IWKID)
```

where IWKID is the GKS workstation identifier of the workstation. If you want the escape element to apply to the next workstation created, you can specify "-1" for the workstation identifier in the **NGSETI** call.

- Private X Color Model

  Sometimes you might need to guarantee that 256 distinct colors are available to a given X window. If an X Workstation is directed to use the Private X Color Model, then that X Workstation will install its own X Color table for the X window. This means that X window will have a different Color table than all the other windows on the screen. Therefore, you usually have to have the mouse pointer in the window for the correct color table to be installed. One disadvantage of this Model is that there is usually a color flashing effect on the screen, since a different color table will be installed for all the other windows on the screen.

  For example, if you want an X workstation to use a private color table, you would make the following call:

  ```
  CALL NGSETI('PC',IWKID)
  ```

  where IWKID is the GKS workstation identifier of the workstation. If you want the escape element to apply to the next workstation created, you can specify "-1" for the workstation identifier in the **NGSETI** call.

- Mixed X Color Model

  The Mixed X Color Model attempts to take the best of both of the previous models. It starts out behaving like the Shared X Color Model, in that it uses the default color table for the screen. It differs in that, once it can't allocate any more colors from the default color table, it allocates its own private color table and starts using it. This way, color flashing is only present if it absolutely needs to be so that the X Workstation can display the correct color. The Mixed X Color Model is the default for Workstations of type 8.

  For example, if you want an X workstation to use the shared color table until it needs to install a private color table, you would make the following call:

  ```
  CALL NGSETI('MC',IWKID)
  ```

  where IWKID is the GKS workstation identifier of the workstation. If you want the escape element to apply to the next workstation created, you can specify "-1" for the workstation identifier in the **NGSETI** call.

Note: It is not possible to switch to a Shared X Color model once a private color table has been installed. This will generate error -221.

Percent Color Error allowed

Percent Color Error is only used if the workstation is using a Shared X Color Model. Escape -1400 is used to set the Percent Color Error allowed. It is used to determine the amount of color error that is allowed before an error message is reported. The way this error is calculated is described above with the Shared X Color Model. Values of 0 and 100 are used to indicate that no error messages are wanted. The error generated is -220.

For example, suppose you want to specify that the percentage error to allow in the color matching algorithm for output to an X window is 20%. The following code would accomplish this:

```
PROGRAM DEMO
C
C  Open GKS, open and activate an X output workstation.
C
      CALL GOPKS(6,IDUM)
      CALL GOPWK(3,2,8)
      CALL GACWK(3)
C
C  Specify a color matching percentage error of 20% for output
C  to an X window workstation with identifier 3.
C
      CALL NGSETI('WO',3)
      CALL NGSETI('PE',20)
C
C  Define colors
C
      CALL GSCR(3,0,0.,0.,0.)
      CALL GSCR(3,1,1.,1.,1.)
      CALL GSCR(3,2,1.,0.,0.)
      .
      .
      .
```

Window size

When you use NCAR GKS to draw to an X11 window, in the default case an X11 window of default size will be created at the time a workstation of type 8 is opened. Before drawing in this window you will need to position it and click on the window.

If you would like to change the size of the window, you will need to put a line of the form:

```
xgks*geometry:  612x612-13-292
```

in your .Xdefaults file. In this case a window of the specified size and position will pop up and drawing into the window will proceed without your having to position the window or click on it.

Obsolete

Escape -1401 is obsolete because it has been replaced by -1402.

PostScript Output

What is PostScript?

Language

PostScript is a general-purpose programming language that contains a rich set of graphics operators. PostScript was designed and developed by Adobe Systems Incorporated; it began to be used by the general public in the mid 1980s. Since that time, the language has gained wide usage as a page description language. PostScript output is produced by many popular word processing and graphics packages and can be displayed on a wide variety of printers, plotters, and workstation screens.

PostScript is an interpreted language rather than a compiled language like Fortran or C. Just like Fortran or C, however, PostScript programs are written as ASCII text files. To display PostScript requires that the program be passed through an interpreter that translates the high-level device-independent PostScript operators into a low-level raster data format for a particular output device.

PostScript is a stack based language and, for the Fortran or C programmer, it can take a bit of getting used to. Fortunately, most users will never need to program in PostScript directly, since user applications generate PostScript automatically. Generating PostScript from NCAR GKS is one good example of automatic PostScript generation.

PostScript fully integrates text and graphics. In fact, text characters are viewed as graphic objects in PostScript. Additionally, PostScript has operators for representing raster images in the language.

Fonts

Any PostScript interpreter must support at least thirteen fonts: the Times family (regular, bold, italic, and bold-italic); the Courier family (regular, bold, oblique, and bold-oblique); the Helvetica family (regular, bold, oblique, and bold-oblique); and a symbol font containing lots of math symbols among others. Some interpreters support dozens or even hundreds of fonts.

File formats

There are some specially-formatted PostScript files (EPS and EPSI files) that you should know about.

EPSF stands for "Encapsulated PostScript File". EPSF is a subset of regular PostScript. The acronym "EPS" stands for "Encapsulated PostScript". You will frequently see reference to "EPS file" or "EPS files" in the literature. EPS files are meant primarily for importing into applications that import PostScript. One primary restriction on EPS files is that they contain only a single page. Another important restriction is that an EPS file must have a comment line that gives the extent of the marks (the "bounding box" of the picture) on the PostScript page that follows. This is so the importing program can figure out where to put the plot. There are certain restrictions on PostScript operators that cannot be in EPS files, like restoring the clipping path to its default value. But these excluded operators are a small subset of the PostScript language that are not frequently used.

EPSI stands for Encapsulated PostScript Interchange format. EPSI files are EPS files that have a "preview bitmap" that represents the PostScript image contained in the file. This bitmap (and it is a bitmap and not a color map) can be used by an importing application to display quickly a picture of the imported file. This is used by applications that do not have a built-in PostScript interpreter. For example, FrameMaker will display this bitmap if it is in the imported file. If you save a FrameMaker document in PostScript format, the original imported PostScript file will be included in the FrameMaker document and you will get the precise picture that you started with. The Adobe manuals describe a portable hex format for the preview bitmap. Some applications want to see a specific type of bitmap in the preview section of an EPSI file, like a Sun raster image, or a GIF file, but how these are handled is vendor-specific.

Recent revisions, level 2

In 1990 a revised version of the PostScript language was released by Adobe Systems. This revision is known as "PostScript Level 2." One of the major additions to the PostScript language was an operator to allow for display of color images, rather than just grayscale images as in PostScript Level 1. Many level 1 PostScript interpreters contain certain level 2 extensions.

References

Many books have been written to describe the PostScript language and how to use it. One of the best books for learning the language remains the *PostScript Language Tutorial and Cookbook* (ISBN 0-201-10179-3) written by Adobe Systems and published by Addison-Wesley in 1985.

If you are going to be doing any work with PostScript beyond using it as an output format, the book *PostScript Language Reference Manual, second edition* (ISBN 0-201-18127-4) is essential. This book was written by Adobe Systems and was published by Addison-Wesley in 1990. The book updates the original reference manual (it includes all Level 2 operators) and contains a wealth of information on the PostScript language.

Producing PostScript Output from NCAR GKS

GKS Workstation Types for PostScript output

PostScript can be produced from the NCAR GKS package in a generic format, or in EPS (Encapsulated PostScript) format, or in EPSI (Encapsulated PostScript Interchange format). The output can be produced in either portrait mode (pictures displayed along the width of the paper or viewing window) or landscape mode (pictures displayed along the length). Also, an option exists for producing color pictures or monochrome pictures. In monochrome output, all colors, except the background color, are mapped to the foreground color. A monochrome plot will not look the same as a color plot viewed on a grayscale printer. The color plot viewed on a grayscale printer will have the colors mapped to shades of gray, whereas a monochrome plot will have only a single foreground color. If you are plotting a color image on a grayscale printer and find the results unsatisfactory since colors are getting mapped to light shades of gray that are difficult to see, it may be better to produce the picture in monochrome.

The various PostScript output options are implemented in NCAR GKS as different workstation types. For an overview of workstation types in NCAR GKS, see the Workstation Functions documentation module.

The following table summarizes the PostScript workstation types in NCAR GKS:

Table 2: NCAR GKS Workstation Types for PostScript Output

Workstation type	orientaton	color/ monochrome	PostScript format
20	portrait	color	ps
21	portrait	color	eps
22	portrait	color	epsi
23	portrait	monochrome	ps
24	portrait	monochrome	eps
25	portrait	monochrome	epsi
26	landscape	color	ps
27	landscape	color	eps
28	landscape	color	epsi
29	landscape	monochrome	ps
30	landscape	monochrome	eps
31	landscape	monochrome	epsi

Unless you want to produce a single picture for importing into some application, such as producing an illustration to import into a word processing package, you most likely will want to select generic PostScript as your output choice. In particular, keep in mind that the EPS and EPSI formats allow only a single picture in them. If you are producing output files with multiple pictures, then you must select the "ps" format.

Output file names

The default file names of PostScript output from NCAR GKS are of the form gmetaXX.YY, where "XX" is the workstation ID used in the GOPWK call and "YY" is either "ps", "eps", or" epsi", as appropriate.

As an example, let’s look at a simple Fortran code fragment that opens and activates a workstation that produces a color EPS file in landscape mode:

```
      CALL GOPKS(3,IDUMMY)
      CALL GOPWK(3,0,27)
      CALL GACWK(3)
```

After this code is executed, a file named "gmeta3.eps" will have been produced that contains the desired PostScript output. It is possible to override the default file name .

Utility Function for Selecting PostScript Workstation Types

In an attempt to make it possible to select the correct PostScript workstation type for the final argument of GOPWK without having to consult the table, a Fortran function

```
      INTEGER FUNCTION NGPSWK(PSTYPE, ORIENT, COLOR)
```

has been written.

All three arguments to this function are CHARACTER variables. The first argument can be one of ’PS’, ’EPS’, or ’EPSI’; the second argument can be one of ’PORTRAIT’ or ’LANDSCAPE’; the final argument can be one of ’COLOR’ or ’MONOCHROME’. In specifying the arguments, only enough characters need be entered so that the values can be differentiated. For example, to specify color, using ’C’ as the third argument would be sufficient. Either upper case or lower case is accepted. Using this function, we could equivalently write the code fragment above as:

```
      CALL GOPKS(3,IDUMMY)
      CALL GOPWK(3,2,NGPSWK('EPS','LAND','COLOR'))
      CALL GACWK(3)
```

Simultaneously Active PostScript Workstations

You can open up to a maximum of fifteen PostScript workstations simultaneously. These workstations can be of any of the available types and can be moved in and out of the active state at any time while they are open.

Note on EPSI output

As of this writing, the preview bitmap contained in EPSI files produced from NCAR Graphics does not represent the picture contained in the EPSI file, but rather is simply a bitmap containing the message: APPLICATION CANNOT DISPLAY NCAR GRAPHIC HERE. VIEW THE POSTSCRIPT OUTPUT TO VERIFY SUCCESSFUL IMPORT. This message is produced on a gridded background. The reason for this situation is that it is not an easy thing to obtain a bitmap of a given PostScript page in a portable manner.

If you are running on a system where ghostscript (Copyright (C) 1990, 1992 Aladdin Enterprises) is installed, then to get an appropriate preview bitmap in your EPSI file, you can use the utility "ps2epsi" that comes with the 2.6.1 release of ghostscript. Ghostscript has been ported to many systems and is distributed under the terms of the GNU Library General Public License. Many applications that import EPS files do not have a built-in PostScript interpreter, they simply display the preview bitmap contained in the imported file on the screen. However, the original PostScript in the imported file is contained in PostScript output from the application, so that when you print the PostScript you will see your original illustration in the file as desired.

Accessing PostScript fonts

Direct assess to PostScript fonts (rather than accessing the stroked or filled characters of the NCAR Graphics package) is now possible via the NCAR GKS PostScript workstations and the GKS function GTX. A translation has been effected for most of the fontcap-defined fonts and filled fonts available via the Plotchar package. PostScript source documentation for Plotchar is available. By selecting fonts with the GKS function GSTXFP and using GTX, you will get PostScript fonts in the resulting output file. These will not be stroked or filled representations of the characters, but characters to be rendered by the PostScript interpreter in use.

There are equivalent PostScript characters for all of the characters from the filled fonts in Plotchar, except for a couple of the math symbols, and except for all of the WMO weather symbols. Additionally, the italic and bold italic fonts, not available in Plotchar, *are* available for the Helvetica, Times, and Courier font families.

There are no equivalents for all characters in all fontcap-defined fonts. For example, the script fonts and the old English font are entirely missing. Also things like the Zodiacal symbols from the NCAR Symbol Font are not available. If a requested character is not available from a PostScript font, NCAR GKS issues a warning message and suggests using Plotchar.

There are two big advantages to using PostScript fonts in place of Plotchar fonts. One is that there is a dramatic difference in PostScript file sizes. Plotchar inserts either strokes or filled areas into the output stream in order to render its characters. There may be dozens of coordinates (with each coordinate pair requiring a dozen bytes in the output PostScript) issued for a single character produced by Plotchar. Using the PostScript fonts requires using only a few bytes additional to those in the string to be rendered.

The second advantage to using PostScript fonts is that they will be rendered accurately at small sizes on low resolution output devices. A great deal of effort has gone into most PostScript interpreters to accurately render characters at all sizes. When the characters produced by Plotchar get too small, some of the areas may fall between pixels on the output device and do not get plotted.

The following table summarizes the relationships between the PostScript fonts and the NCAR GKS fonts:

Table 3: Mapping between NCAR GKS fonts and PostScript fonts

NCAR GKS font number	NCAR GKS font name	PostScript equivalent
1	default	Helvetica
-2	Hershey Cartographic Roman	Helvetica (scaled small)
-3	Hershey Cartographic Greek	Greek from Symbol font (scaled)
-4	Hershey Simplex Roman	Helvetica
-5	Hershey Simplex Greek	Greek from Symbol font
-6	Hershey Simplex Script	none
-7	Hershey Complex Roman	Times-Roman
-8	Hershey Complex Greek	Greek from Symbol font
-9	Hershey Complex Script	none
-10	Hershey Complex Italic	Times-Italic
-11	Hershey Complex Cyrillic	none
-12	Hershey Duplex Roman	Helvetica
-13	Hershey Triplex Roman	Times-Bold
-14	Hershey Triplex Italic	Times Bold-Italic
-15	Hershey Gothic German	none
-16	Hershey Gothic English	none
-17	Hershey Gothic Italian	none
-18	Hershey Math Symbols	math symbols from Symbol font
-19	Hershey Symbol Set1	none
-20	Hershey Symbol Set2	none
-21	NCAR Helvetica	Helvetica
-22	NCAR Helvetica-Bold	Helvetica-Bold
-23	none	Helvetica-Oblique
-24	none	Helvetica-BoldOblique

```
  -25        NCAR Times-Roman            Times-Roman
  -26        NCAR Times-Bold             Times-Bold
  -27        none                        Times-Italic
  -28        none                        Times-BoldItalic
  -29        NCAR Courier                Courier
  -30        NCAR Courier-Bold           Courier-Bold
  -31        none                        Courier-Oblique
  -32        none                        Courier-BoldOblique
  -33        NCAR Greek                  Greek from Symbol font
  -34        NCAR Math Symbols           math symbols from Symbol font
  -35        NCAR Text Symbols           text symbols from Symbol font
  -36        NCAR Weather 1              none
  -37        NCAR Weather 2              none
------------------------------------------------------------------------
```

Cell array translation

GKS Cell Arrays are mapped into the "image" or "colorimage" operator in PostScript output from NCAR GKS. PostScript level 1 interpreters are only required to handle gray scale images, whereas PostScript level 2 allows for color images via the "colorimage" operator. What is put into a PostScript file resulting from a Cell Array in GKS is PostScript code that will check if the given PostScript interpreter supports "colorimage"; if it does, then a color image will be produced, and if it does not, then a grayscale image will be produced where the colors in the cell array are mapped to grayscale in accordance with the NTSC (National Television Standards Committee) equation: *Intensity = 0.30*red + 0.59*green + 0.11*blue.*

Producing PostScript images from GKS cell arrays represents a significant savings in file sizes compared to producing an ncgm file and converting the ncgm file to PostScript via ctrans. The ctrans route draws a filled area for each cell in the cell array and the resultant PostScript file will, in general, be more than ten times larger than creating the PostScript file directly from NCAR GKS.

GKS ESCAPE functions controlling PostScript Output

In what follows, details are given on the escape functions in NCAR GKS that apply specifically to PostScript workstations. These escape functions control aspects of the PostScript output that are language-specific and lie outside the scope of the GKS Standard. The user interface to these escape functions is via the entries NGSETI, NGSETR, and NGSETC.

Positioning PostScript output on the page

The GKS Standard requires that the aspect ratio of a plot be preserved in its mappings from world coordinates to normalized device coordinates (NDC) to device coordinates. In particular this means that in mapping the default NDC unit square onto the display surface of the output device it should be mapped onto a square. This prevents things like circles being turned into ellipses and squares being turned into rectangles. In the default case, PostScript output is placed on the output page in a square that is centered on the page.

On certain occasions it may be desirable to use the entire rectangular area of the output page, or even to change the aspect ratio of the plot. PostScript output from NCAR GKS can be made to occupy any position on the page by way of setting coordinate boundaries via GKS escape function -1521. The integer variables that control this are LX, LY, UX, UY. These numbers specify corner points (LX,LY) and (UX,UY) that bound the output on the page. The numbers are specified in the default PostScript coordinate space where one unit corresponds to 1/72 of an inch. This means that an 8.5" x 11" page in portrait mode would be addressed from (0,0) in the lower left corner to (612,792) in the upper right corner. In order to preserve aspect ratio, the output rectangle should be specified as a square. For example, if you want your output to occupy a square in the upper left part of a page, you would make calls such as:

```
CALL NGSETI('LX',50)
CALL NGSETI('LY',500)
CALL NGSETI('UX',300)
CALL NGSETI('UY',750)
```

Values must be specified for all four parameters LX, LY, UX, and UY.

Using the full page

A common desire is to use a full printed page to display pictures. As discussed earlier, unless you can live with the undesirable results of simply scaling your plot to fill the entire page, you will need to restrict your original plot to using an aspect ratio that is compatible with the output page (11/8.5). To produce output on a full page, you will probably want to chose one of the landscape mode workstations. In the default case, a plot in landscape mode will be exactly what is was in portrait mode, except rotated clockwise by 90 degrees. Using the positioning parameters LX, LY, UX, and UY requires some care in landscape mode and requires that you understand how landscape mode is implemented in the PostScript workstations. In landscape mode the parameters LX, LY, UX, and UY specify default PostScript coordinates (one unit equals 1/72 of an inch) that define the corner points (LX,LY) and (UX,UY) of a rectangle. The plot will be positioned within this rectangle, then the rectangle will be rotated by 90 degrees about the point (LX,LY) and then translated vertically by an amount so that the lower left corner of the rotated plot will coincide with the original lower left corner (LX,LY). In specifying the values for LX, LY, UX, and UY realize that, if the final plot is to fill the page in landscape mode, the value specified for UX would run off the right of an 8.5" x 11" page if plotted in portrait mode.

For example, suppose that you choose a landscape mode PostScript workstation and specify the following values:

```
LX = 50
LY = 50
UX = 742
UY = 562
```

The plot will originally be scaled to fit in the rectangle with lower left corner (50,50) and upper right corner (742,562) [if drawn in portrait mode, this plot would run off the right of an 8.5" x 11" piece of paper]. This rectangle is then rotated by 90 degrees clockwise about the point (50,50), giving a rectangle with coordinates (50,50) at the upper left and (562,-642) at the lower right. Translating this rectangle vertically by 692 gives a rectangle with corner points (50,50) at the lower left and (562,742) at the upper right. Since the extreme corners of a printed 8.5" x 11" page are (0,0) and (612,792), the above rectangle will fit entirely within the page. The above settings distort the aspect ratio of the plot by elongating it and then rotating it so that it fits on the page.

You will probably not want to distort the aspect ratio of your plots when plotting to a full page, since the plots will have characters that look funny, circles will appear as ellipses, and so forth. In this case you should start with a plot that has the aspect ratio of a printed page (8.5/11) and scale it appropriately to fit the page. Figure 19 and Figure 20 illustrate this. The landscape version in Figure 20 was positioned using workstation type 28 (color EPSI in landscape mode) and making the following calls:

```
CALL NGSETI('LX',50)
CALL NGSETI('LY',50)
CALL NGSETI('UX',742)
CALL NGSETI('UY',742)
```

Notice that the aspect ratio of the plot being approximately 3 to 4 requires that white space be left above the plot in order for it to be plotted correctly. See the above discussion on positioning your output in order to understand the above subroutine calls.

Figure 19 - Plot in portrait mode

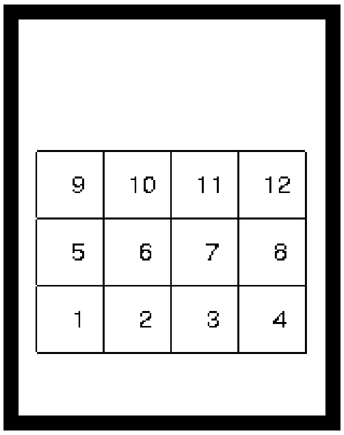
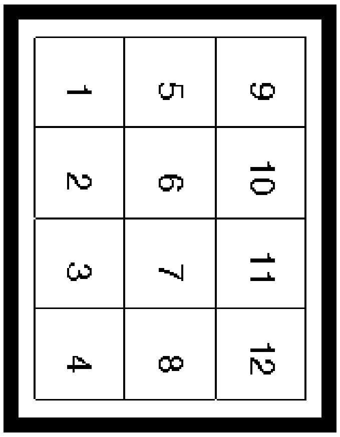


Figure 20 - Plot from Figure 19 scaled to fill the page in landscape mode



Notes on page positioning parameters:

- If you are setting LX or UX, be sure to set LX before UX since NGSETI checks to see that UX > LX when UX is set. Similarly for LY and UY.
- Since the coordinate positioning has to be known when the output PostScript file is initialized, the settings of LX, LY, UX, and UY apply to the first PostScript workstation opened with GOPWK after the parameters are set. And they will apply only to that workstation, since they are reset to their defaults (LX=36, LY=126, UX=576, UY=666) after GOPWK is called. To reiterate, settings for LX, UX, LY, and UY must be made *before* the workstation is opened to which they apply, but after GKS has been opened.

Some experimenting with various settings for the workstation types and values for LX, LY, UX, and UY should give you a feeling for how they work.

Overriding Default File Names

Names for the PostScript output files can be assigned to override the default names. The name for the output file must be known at the time the workstation is opened with GOPWK. Making a call:

```
CALL NGSETC('ME','postscript_file_name')
```

just prior to the GOPWK call that opens the workstation will assign the desired name to the output file. The name can be up to 256 characters long.

You can assign 'stdout' to the output file name with a call to NGSETC and the output will go to standard out. This is only practical when you are writing to a single PostScript output file. Also, be aware that errors or warnings are usually written to the standard output file and intermingling these with the output PostScript will corrupt the PostScript.

Selecting a color model

By default, PostScript produced from NCAR Graphics uses the RGB color model. You can make the PostScript output use the CMYK color model (particularly useful in printing) by making a call

```
CALL NGSETI('CM',IVAL)
```

where IVAL=0 (IVAL=1 selects the default RGB model).

The above call must be made *just prior to* the GOPWK call that opens the PostScript workstation.

Background color.

Just as for output to NCGM or X11 windows, the value for GKS color index zero controls the background color setting for PostScript output. By default, the background color is applied to the normalized device coordinate viewport. If you want to have the background color apply to the entire output page for a given PostScript workstation, make the calls:

```
CALL NGSETI('Workstation',WKID)
CALL NGSETI('Full background',1)
```

where WKID is an integer workstation identifier. The setting for full background will apply only to the workstation specified in the most recent NGSETI call that sets the workstation ID.

Line style attributes

PostScript has several line style attributes that are not part of GKS. Control over these attributes can be obtained using NGSETI calls. Figure 21 illustrates all of the line style attributes discussed in this section.

Line joins

The line join type controls the appearance of how line segments are connected. There are three possible values for the line join parameter: 0 for miter joins; 1 for rounded joins; 2 for bevel joins. For mitered joins, the edges of the line segments are extended until they meet. This join is also affected by the miter limit. For rounded joins, the line segments are joined with a circle at the join center where the circle has radius equal to the line width. For bevel joins, the line segments are drawn with butt ends (see the discussion of line caps) and the notch at the larger angle between the segments is filled with a triangle. Rounded joins are the default for PostScript produced from NCAR Graphics. Values for line join can be set by making the appropriate NGSETI call. See Figure 21 for an example.

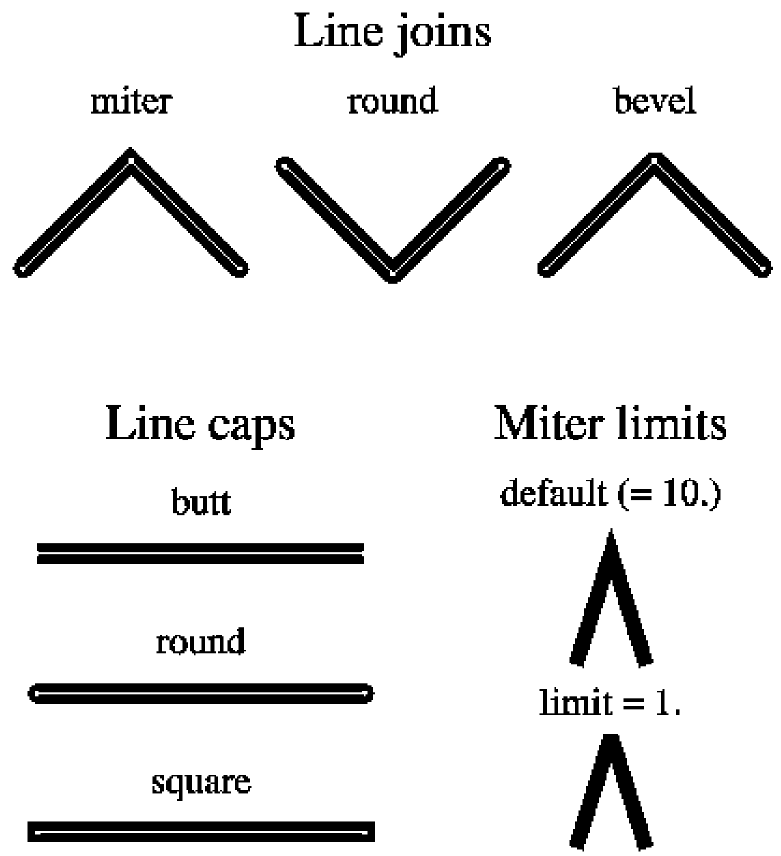
Line caps

The line cap attribute controls how the ends of line segments are stroked. There are three possible values for the line cap parameter: 0 for butt caps; 1 for round caps; 2 for projecting square caps. For butt caps, the line segments are terminated with square ends perpendicular to the line path. For round caps, the line segments are terminated with semicircles centered at the line endpoint and having a radius equal to the line thickness. Projecting square caps are the same as butt caps except the cap is extended beyond the line end points by half a line width. Round caps are the default for PostScript produced from NCAR Graphics. Values for line cap can be set by making the appropriate NGSETI call. See Figure 21 for an example.

Miter limits

If, for mitered line joins, line segments meet at a very acute angle, then the mitered join can result in a spike that extends significantly beyond the intersection of the segments. To prevent this, PostScript switches from a mitered join to a beveled join at an angle that is controled by the miter limit parameter. The smaller the miter limit is, the less tolerant PostScript is of exaggerated spikes at line joins. The default setting for the miter limit is 10. and the smallest the miter limit can ever be is 1. Setting the miter limit to 1. will essentially result in beveled line joins; setting the miter limit to values much higher than 10. will result in long spikes at lines joined at acute angles. Values for the miter limit can be set by making the appropriate NGSETI call.

Figure 21 - Line style attributes



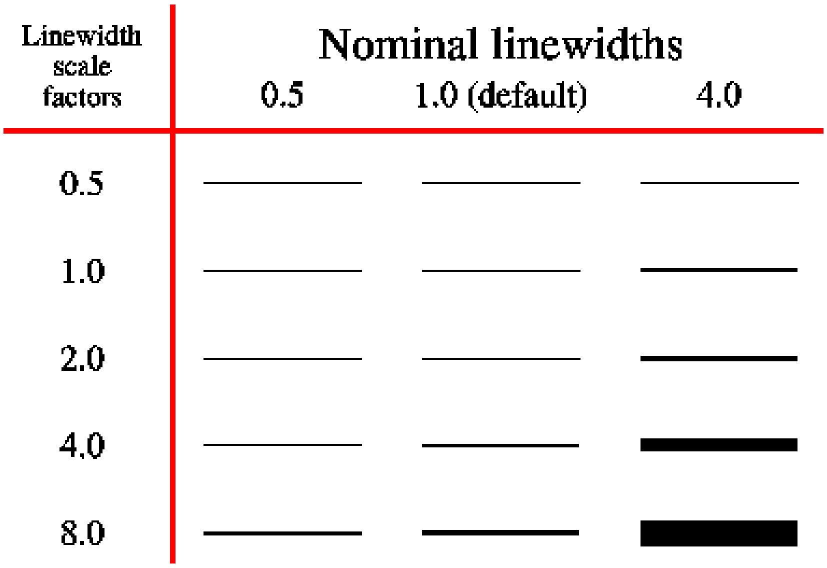
Line widths

The GKS call GSLWSC can be used to set relative linewidths. The scale factor supplied to GSLWSC is applied to a default nominal line width. This nominal line width is implementation specific. You can control the size of this nominal linewidth for PostScript output by using the "NO" (for "Nominal linewidth") parameter in a call to NGSETR. For example, the calls:

```
      CALL NGSETI('WO',WKID)
      CALL NGSETI('NO',2.0)
```

would double the width of the nominal linewidth. This means that all lines would uniformly appear to be twice as thick as they would be in the default case. Figure 22 illustrates the result of changing the size of the nominal linewidth and then changing linewidths using GSLWSC. Just remember that the linewidth scale factors given to GSLWSC apply to the nominal linewidth, whereas setting a new value for the nominal linewidth will uniformly change the width of all lines.

Figure 22 - Line widths



Size of the operand stack

As pointed out in the introduction to this documentation module, PostScript is a stack based language. Among other things, this means that all operators operate on elements that have been pushed onto the operand stack. The size of this stack is implementation dependent, but it must be able to contain at least 500 elements at the minimum. Most PostScript interpreters have a bigger operand stack than this.

The relevance of this value for PostScript output from NCAR GKS is the size of the output files that contain numerous filled areas. If all of the coordinates of a filled area can be pushed onto the stack, then a locally-defined macro can be utilized to save space and interpretation time. The savings here is not great, about 10% at most, so unless you are really concerned about file sizes and efficiency, resetting the size of the operand stack should be of little concern to you.

Also, keep in mind that you will lose portability by resetting this parameter. If you print the file using a PostScript interpreter that has a smaller operand stack than what is specified, then you will get an error from the PostScript interpreter. Changing the limit on the PostScript stack size can be effected by using the "ST" (or "Stack size") parameter in an NGSETI call.

Maximum number of points in a path

In drawing lines and filled areas, PostScript builds a "path" and then strokes it out. The default limit on the maximum number of points in a path is 1500 (this limit is set to 1300 by default in PostScript output from GKS to be on the safe side). Polylines generated by NCAR GKS that exceed this limit are broken into paths that fall within the limit.

Filled areas present a more difficult situation in that the entire path must be presented in its entirety to be filled. For filled areas that exceed the limit on the maximum path size, software fill is utilized. This means the instead of presenting an entire path to the PostScript fill operator, the area is filled by drawing a sequence of horizontal lines that are thick enough so that no gaps will be displayed. If you are generating filled areas that have more than 1300 points in them and you are using a PostScript interpreter that allows for more points than this, then you may want to consider increasing the limit on the maximum number of points in a path. You can do this by using the "PA" (or "Path size") parameter in an NGSETI call.

The savings in file size and interpretation time can be substantial (up to 50% or greater for files containing only large filled areas), but on the average (files that have one or two large filled areas) the savings are considerably more modest (1-10%). Also remember that you will lose portability by resetting this parameter. If you print the file using a PostScript interpreter that has a smaller path size than what is specified, then you will get an error from the PostScript interpreter. Setting the maximum path size to a small number can be used to force software fill, if that is ever desired.

Specifying spacing between software fill lines

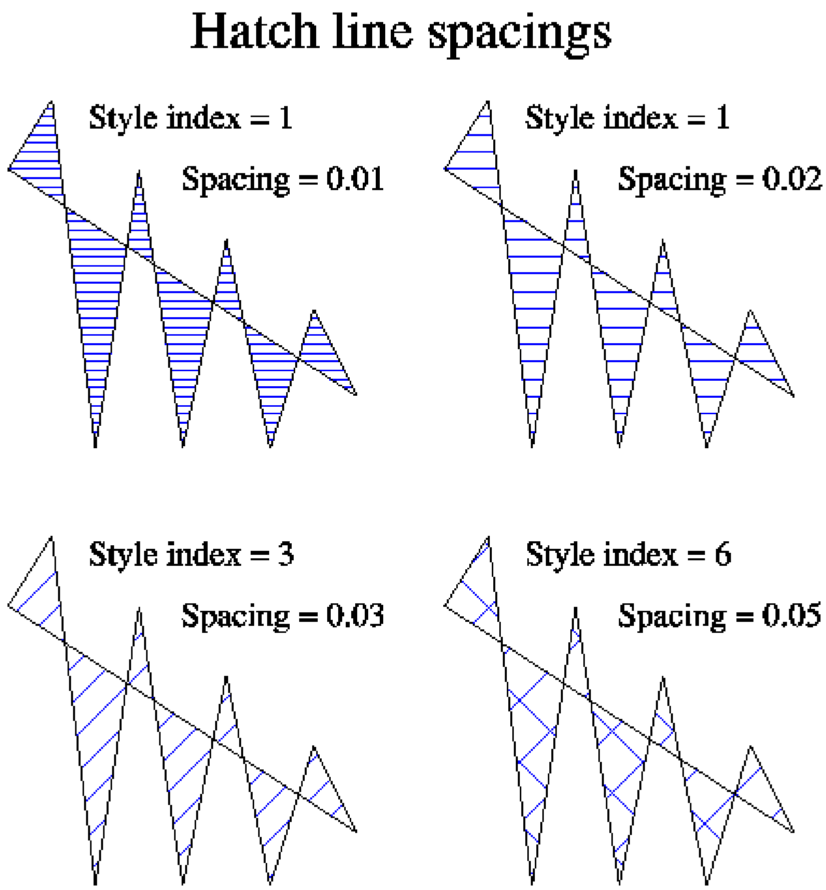
When a filled area is requested that has more than the maximum number of points in it than are allowed in a PostScript interpreter (1300 in PostScript output from NCAR GKS), software fill is implemented. This amounts to drawing a sequence of horizontal lines that will fill the given area. These lines are chosen to have a thickness so that no gaps result between fill lines. The distance between fill lines is specified as a real number in the range 0. to 1. (normalized device coordinate space). By default this value is .0005 which will result in 2000 lines being used to fill the largest possible area.

If you are plotting on a device with crude resolution and could get by with many fewer lines than the default, then you may want to consider resetting this value, which can be done by using the "FI" (or "Fill lines") parameter in an NGSETR call. If you are going to a super high resolution device, then it is possible that you will see discrete line ends on edges of filled areas that are at an acute angle. If this is undesired, then you may consider decreasing the value for FI (increasing the number of fill lines). Remember that by tuning the number of fill lines for a specific device may cause undesirable results on another device. Also, remember that software fill is not invoked unless the number of points in a filled area exceeds the maximum number of points allowed in a path.

Specifying spacing between hatch pattern lines

The spacing between lines composing a hatch fill pattern can be controlled with the NGSETR parameter "HA" (or 'Hatch spacing'). This applies only when the fill area interior style is set to 3 (see GSFAIS). The fill spacing is specified as a real number between 0. and 1. , inclusive (normalized device coordinate space).

Figure 23 - Hatch line spacings



Specifying the user coordinate space resolution

The default user coordinate space in PostScript locates the origin at the lower left corner of the output page; the positive X axis extends horizontally to the right and the positive Y axis extends vertically upward. The length of a unit along either axis is 1/72 of an inch. Coordinates can be specified as either integers or real numbers.

In the default user integer coordinate space, an 8.5" x 11" output page in portrait mode would have X coordinates in the range 0 to 612 and Y coordinates in the range 0 to 792.

This is a fairly crude resolution, and when it is mapped onto a device space with much higher resolution (such as a 300dpi printer), the increased resolution of the output device is not fully utilized. PostScript allows for scaling the user space so that you can use higher resolutions. The default user coordinate space in PostScript produced from NCAR GKS scales the normal default user space by 25. For an 8.5" x 11" page in portrait mode this produces coordinates in the range 0 to 15300 in X and 0 to 19800 in Y. This space is the default user space scaled by 25. This scale factor can be controlled by the NGSETI parameter 'CO' (or 'Coordinate scale'). By default it is 25 as discussed above. If you are plotting to a device with crude resolution, then you might consider setting this scale factor to something smaller than 25. By doing so, the file sizes will be reduced in accordance with the smaller number of characters needed to represent the coordinates in the resultant PostScript file. If you are plotting to a very high resolution printer and notice that you are not taking full advantage of the resolution of the device, then you may want to consider increasing the resolution of the PostScript user space by setting the scale factor controlled by 'CO' to something larger than 25.

Links: GKS Index, GKS Home

SPPS Functions

The NCAR System Plot Package Simulator (SPPS) was written for two purposes. One was to smooth the transition from the pre-GKS version of NCAR Graphics to the GKS package. Toward this purpose many subroutine entries were implemented by making the appropriate GKS calls.

The second purpose of SPPS was to implement a number of important features which were missing in the GKS standard. In this documentation module are presented some of the helpful features of SPPS that resulted from these two objectives.

SPPS routines that have GKS analogs are repeated here for the sake of convenience. Discussed in the document NCAR Graphics Fundamentals, UNIX Version are the additional SPPS subroutines:

- FRAME - Flush all buffers and advance to a new picture.
- LINE - Draws a line between two points.
- CURVE - Draws line segments through a set of coordinate positions.
- POINT - Draws a point at a given coordinate.
- POINTS - Draws a point or character at a set of given coordinates with an option of connecting them with line segments.

Man pages exist for all of these subroutines. For the C interfaces for SPPS routines, consult the man pages for the equivalent Fortran entry.

Open and Close GKS

The SPPS call CALL OPNGKS is equivalent to the following set of three GKS calls:

```
CALL GOPKS(1,IDUMMY)
CALL GOPWK(1,2,1)
CALL GACWK(1)
```

which open and activate workstation 1, of type 1, on unit 2. This is an NCAR CGM metafile output workstation.

The SPPS subroutine CALL CLSGKS is equivalent to the following set of three GKS calls:

```
CALL GDAWK(1)
CALL GCLWK(1)
CALL GCLKS
```

which deactivate and close workstation 1 (the CGM), and close GKS.

Normalization Transformations

As discussed in the Coordinate Systems module, NCAR Graphics user coordinates provide two major extensions to GKS world coordinates: mirror imaging and log scaling. The normalization transformations which incorporate these extensions are provided by the SET subroutine of SPPS. A single call to SET allows you to define a window, a viewport, mirror imaging, and log scaling. You are encouraged to use the SET call for establishing normalization transformations. Figure 24 and Figure 25 show log scaling and axis reversal, respectively, with SPPS.

```
------------------------------------------------
       Argument  |    Type     | Mode   | Dimension
------------------------------------------------
CALL SET   (XLV, |    Real     | Input  |
            XRV, |    Real     | Input  |
            YBV, |    Real     | Input  |
            YTV, |    Real     | Input  |
            XLW, |    Real     | Input  |
            XRW, |    Real     | Input  |
            YBW, |    Real     | Input  |
            YTW, |    Real     | Input  |
            LS)  |    Integer  | Input  |
------------------------------------------------
```

XLV, XRV, YBV, YTV
    X left, X right, Y bottom, and Y top of the viewport in normalized device coordinates.
XLW, XRW, YBW, YTW
    X left, X right, Y bottom, and Y top of the window in user coordinates.
LS
    Log scaling with options of:

        1 - linear X, linear Y
        2 - linear X, log Y
        3 - log X, linear Y
        4 - log X, log Y

    The default for LS is 1.

Mirror imaging is determined by the ordering of the range of the user coordinates that define the plotting window. The SET call to establish the normalization transformation shown in Diagram 1 is:

```
CALL SET(.15,.95,.10,.90,100.,1000.,100.,1000.,1)
```

In this case, both X and Y have axes that run from 100. to 1000. units; thus, mirror imaging is not in effect on either axis. The SET call to establish the normalization transformation shown in Diagram 2 is:

```
CALL SET(.15,.95,.10,.90,1000.,100.,100.,1000.,2)
```

In this case, values along the X axis run from max to min due to reverse ordering of the window coordinates. Note that arguments 5 and 6 (X left and X right of the window) are now 1000. and 100., rather than 100. and 1000.; thus, mirror imaging is in effect for the X axis. In addition, logarithmic scaling is in effect for the Y axis by setting the last argument to 2.

Inquiry Functions

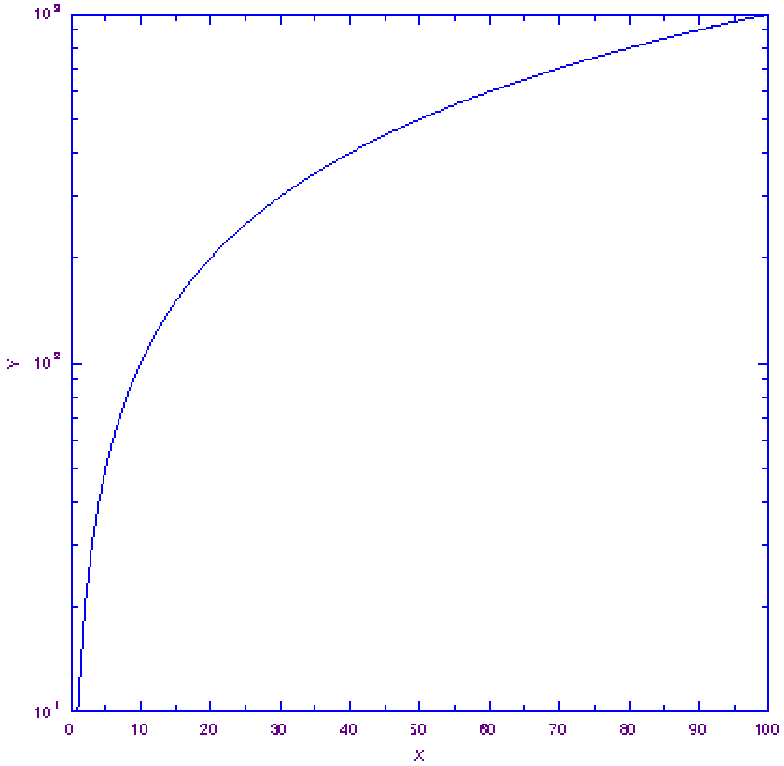
Normalization Transformations

SPPS contains an inquiry function that will return the viewport, window, log scaling, and mirror imaging currently in effect.

```
      CALL GETSET(XLV, XRV, YBV, YTV, XLW, XRW, YBW, YTW, LS)
```

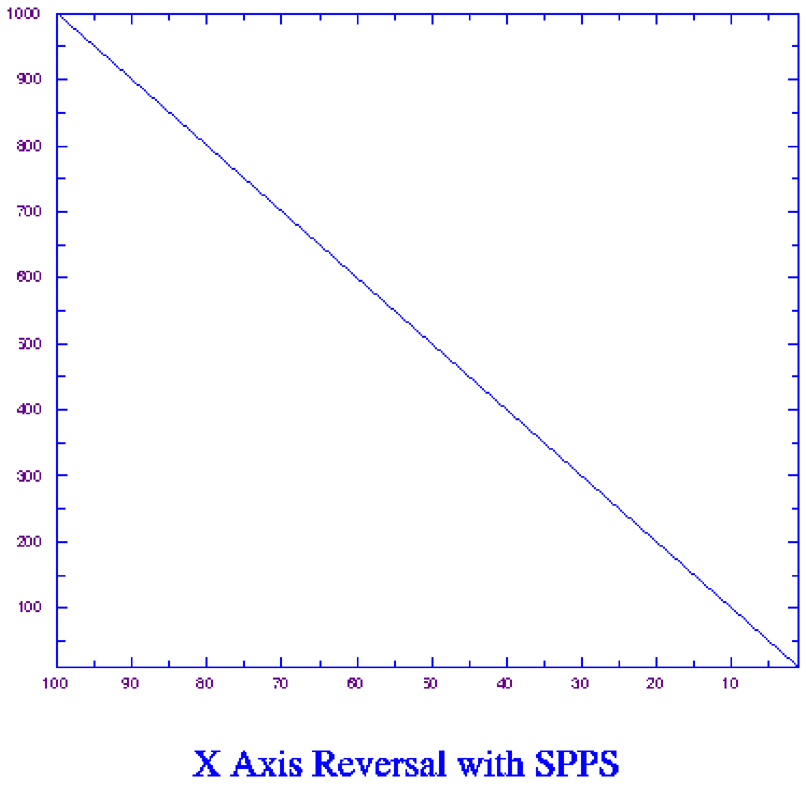
Argument descriptions are the same as for the SET call defined above. They are all output arguments.

Figure 24 - Log scaling



Log Scaling with SPPS

Figure 25 - Mirror imaging



Links: GKS Index, GKS Home

Utility Functions

In addition to the SPPS Functions, there are several Utility Functions that augment the NCAR GKS package. The members of this set functions discussed here are:

- NGCKOP - function to check if a given workstation is open.
- NGDOTS - draws filled dots or circles with any size or color.
- NGMFTC - temporarily close a metafile workstation.
- NGPICT - controls breaks in a picture drawing sequence.
- NGREOP - reopens a metafile for appending.
- NGSRAT - Saves/restores/sets NCAR GKS primitive attribute settings.

NGCKOP

NGCKOP is a function that takes a GKS workstation identifier as an argument and returns a "1" if the workstation is open and a "0" otherwise.

```
------------------------------------------------------
              Argument                |    Type     | Mode
------------------------------------------------------
```

```
INTEGER FUNCTION NGCKOP (WKID)  |  Integer  | Input
-------------------------------------------------------
```

WKID
    Integer GKS workstation identifier.

The statement:

```
IOPEN = NGCKOP(WKID)
```

will set IOPEN to "1" if the workstation whose identifier is WKID is currently open, otherwise IOPEN will be set to "0".

NGDOTS

The GKS standard requires that marker type 1 always be displayed as the smallest possible dot. For this reason a dot marker *will not be scaled*, regardless of the value of the marker size scale factor supplied to GSMKSZ. The NGDOTS utility has been added to allow for drawing dots or circles at any size and in any color.

```
-------------------------------------------------------
  Argument             |   Type     | Mode   | Dimension
-------------------------------------------------------
CALL NGDOTS  (X,       |   Real     | Input  | NUM
              Y,       |   Real     | Input  | NUM
              NUM,     |  Integer   | Input  |
              SIZE,    |   Real     | Input  |
              ICOLOR)  |  Integer   | Input  |
-------------------------------------------------------
```

X
    Specifies the X world coordinates where a sequence of circular filled dots (or circles) is to be drawn.
Y
    Specifies the Y world coordinates where a sequence of circular filled dots (or circles) is to be drawn.
NUM
    The number of dots (or circles) to be drawn.
SIZE
    Specifies the size, in world coordinate Y-axis units, of the diameter of each dot to be drawn.
ICOLOR
    The GKS color index specifying what color the dots will be.

The coordinates must be world coordinates and not user coordinates. NGDOTS does not respect the log scaling or axis reversal options of the SET call and will report a warning if these are not set to their default values.

The dots are scaled appropriately so that they will be circular when the normalization transformation does not preserve aspect ratio.

The algorithm is to construct a single circle and then translate it to the various coordinate positions. The original circle is computed by using trig functions to get points in the first octant, and then using symmetries to get the rest of the points on the circle.

The number of points used for the circle is adjusted depending on the relative size of the circle. The maximum number of points is 512 and the minimum is 8.

If circles are desired rather than dots, then toggle the ngmisc parameter *CT* (see the man page for *ngmisc_params* which describes all of the available ngmisc parameters) with a call to NGSETI:

```
CALL NGSETI('CT',1)
```

before NGDOTS to get circles, and

```
CALL NGSETI('CT',0)
```

before calling NGDOTS to get dots. The default is to draw dots.

Assuming that normalization transformation 0 is in effect and that color index 2 has been defined as red in a call to GSCR, then

```
CALL NGDOTS(0.5, 0.25, 1, 0.1, 2)
```

would draw a single red dot at world coordinate position (.5,.25) with a diameter of 0.1 .

Assuming that normalization transformation 0 is in effect and that color index 2 has been defined as red in a call to GSCR, then

```
CALL NGDOTS(0.5, 0.25, 1, 0.1, 2)
```

would draw a single red dot at world coordinate position (.5,.25) with a diameter of 0.1 .

Do an "ncargex fngngdts" for a relevant example.

NGMFTC

NGMFTC temporarily closes an NCGM, allowing for subsequent reopening with NGREOP.

Argument	Type	Mode
CALL NGMFTC (WKID)	Integer	Input

WKID
: Integer GKS workstation identifier.

NGMFTC can be called at any time after a metafile workstation has been opened, even in the middle of drawing a picture. The primary use of NGMFTC is to allow for writing to more than one metafile in a single job execution. Since NCAR GKS does not allow for having more than a single NCGM workstation open at a time, you can use NGMFTC to stop output to one metafile while you write to another metafile and then resume output to the initial metafile.

If you simply want to suspend output to a metafile workstation, but do not need to write to another metafile workstation, simply use the GKS entry GDAWK to deactivate the workstation, thus suspending output, and GOPWK to reactivate the workstation when you want to resume output.

For saving the state of all primitive attribute values of an NCGM workstation at the time of the temporary close and restoring them at the time of the reopen with NGREOP, see the documentation for NGSRAT.

If NGMFTC is used and NGREOP is not subsequently invoked to reopen the file so that it can be terminated normally in the same job step, then an improperly terminated metafile will result from the temporary close.

Do an "ncargex pgkex27" for a relevant example.

NGPICT

NGPICT can be utilized to handle breaks in a GKS picture drawing sequence. The actions taken depend on whether the designated workstation is a metafile or an output/input workstation. An option is provided for prompting the user when an output/input workstation is ready and waiting after a pause. NGPICT is designed to be used when more precise control over individual workstations is desired than that offered by FRAME. This might be desirable, for example, when an application is controlling several simultaneously active workstations. To use NGPICT one should be opening GKS workstations with GOPWK rather than with OPNGKS.

```
-----------------------------------------
  Argument            |   Type    | Mode
-----------------------------------------
CALL NGPICT (WKID,    |  Integer  | Input
             ACTION)  |  Integer  | Input
-----------------------------------------
```

WKID
: designates the workstation identifier of a workstation as it was specified in a GOPWK call.

ACTION
: specifies the action to be taken on the workstation WKID. Legal values are:

  0
  : Execute an UPDATE WORKSTATION on WKID. All pending action items for WKID are made current. All buffers are flushed.

  1
  : Execute an UPDATE WORKSTATION followed by a CLEAR WORKSTATION. The workstation screen is cleared after being made current.

  2
  : Execute an UPDATE WORKSTATION followed by a pause waiting for a mouse click or a key click.

  3
  : Execute an UPDATE WORKSTATION followed by a pause followed by a CLEAR WORKSTATION after the pause has been terminated by a mouse click or a key click.

  4

Same as 3 except a "&ltREADY>" prompt is issued in the lower left corner of the window after the UPDATE WORKSTATION.

If WKID designates a metafile, then a CLEAR WORKSTATION is done which inserts an END PICTURE into the metafile. The result produced in this case would be the same as that resulting from a call to FRAME. The only valid actions for a metafile workstation are 0 and 1.

If WKID=2 designates a metafile workstation, then

```
CALL NGPICT(2,1)
```

would duplicate the action of a FRAME call.

If WKID=2 designates an X11 workstation, then

```
CALL NGPICT(2,4)
```

would cause a pause in the X11 window and wait for a key click or a mouse click. A "&ltREADY>" prompt, indicating that the window is waiting for a mouse click or a key click, would appear at the lower left of the X11 window after the window has been updated. After a mouse click in the X11 window, that window will be cleared before program execution continues.

NGREOP

NGREOP is called to reopen an existing NCAR Graphics metafile for appending. The reopened metafile may be a complete metafile that was created in the same job step, or a different job step, or the metafile may be an incomplete one that has been temporarily closed with a call to NGMFTC.

Argument	Type	Mode	Dimension
CALL NGREOP (WKID,	Integer	Input	
CONID,	Integer	Input	
ITYPE,	Integer	Input	
FNAME,	Character	Input	
IOPT,	Integer	Input	
IAT,	Integer	Input	14
RAT,	Integer	Input	7
NCOLRS,	Integer	Input	
NSTART,	Integer	Input	
CTAB)	Real	Input	3 x NCOLRS

WKID
: designates a GKS workstation identifier.

CONID
: designates a GKS conection identifier.

ITYPE
: specifies the workstation type. Currently the only valid value is "1". This subroutine may be augmented to accommodate PostScript files in the future.

FNAME
: specifies the filename of the metafile being opened for appending.

IOPT

specifies the desired action:

0

reestablish the color table only (using the color table in argument CTAB described below). The attributes of the reopened workstation will be set to default values and may well be out of sync with the current GKS attributes. action items for WKID are made current. All buffers are flushed.

1

reestablish the color table and GKS state (the GKS state as supplied in arguments IAT and RAT described below). The state values will not be flushed to the metafile.

2

reestablish the color table and GKS state and flush the GKS state values to the metafile.

3

reestablish the color table and flush the current GKS state values to the metafile (not the values in IAT and RAT).

IAT

contains GKS integer state variables as detailed in the documentation for NGSRAT.

RAT

contains GKS floating-point state variables as detailed in the documentation for NGSRAT.

NCOLRS

specifies the number of colors in the color table supplied in argument CTAB, described below.

NSTART

specifies the color index associated with the first color in the color table CTAB (all other color indices are filled in in sequence). For example, if NCOLRS = 3 and NSTART = 4, then the color values defined in CTAB would be used to define color indices 4, 5, and 6.

CTAB

specifies a color table used to initialize the reopened metafile. This color table does not necessarily have to agree with the color table in effect when the original metafile was created.

The most common usage of NGREOP would be in conjunction with usage of NGSRAT and NGMFTC. NGMFTC would be used to temporarily close a metafile and NGSRAT used to save the state of the GKS primitive attributes at the time of the close. To reopen the temporarily closed metafile you would call NGREOP with the attribute settings previously saved and a setting of IOPT of 2. NGMFTC can be used to temporarily close a metafile any time after that metafile has been opened, even in the middle of a picture.

NGREOP only reopens the metafile and does not activate it. It will be necessary to call GACWK before sending graphics primitives to the reopened metafile.

NGREOP can also be used to reopen a previously created metafile - either created in an independent job step, or in the same job step.

The following sequence:

```
CALL NGMFTC(1)
CALL NGSRAT(2, IAT, RAT)

  ... do stuff
```

```
       CALL NGREOP(CALL NGREOP(1, 2, 1, 'gmeta1', 2,
     +            IAT, RAT, NCOLS, 0, CTAB)
```

would temporarily close the metafile with workstation ID of 1 (in this case a metafile with name "gmeta1") and save the GKS state variables at the time of the close. Then the call to NGREOP would reopen the metafile for appending.

Do an "ncargex pgkex27" for a relevant example.

NGSRAT

NGSRAT can be used to save, restore, or set all NCAR GKS primitive attribute values in a single call.

Argument	Type	Mode	Dimension
CALL NGSRAT (IOPT,	Integer	Input	
IAT,	Integer	Input	14
RAT)	Real	Input	7

IOPT
: specifies the desired action:

  0
  : save the current settings of all attributes internal to the subroutine, i.e. not in the IAT and RAT arrays.

  1
  : restore the context from the most recently saved state.

  2
  : return the current context in the output arrays IAT (integer attributes) and RAT (floating-point attributes) as follows:

    IAT(1)
    : Clip indicator

    IAT(2)
    : Line type

    IAT(3)
    : Polyline color index

    IAT(4)
    : Marker type

    IAT(5)
    : Polymarker color index

    IAT(6)
    : Text font

    IAT(7)
    : Text precision

    IAT(8)
    : Text color index

    IAT(9)

Text path

IAT(10)

Text horizontal alignment

IAT(11)

Text vertical alignment

IAT(12)

Fill area interior style

IAT(13)

Fill are style index

IAT(14)

Fill area color index

RAT(1)

Linewidth scale factor

RAT(2)

Marker scale factor

RAT(3)

Character expansion factor

RAT(4)

Character spacing

RAT(5)

Character height in world coordinates

RAT(6)

Character up vector, X component in world coordinates

RAT(7)

Character up vector, Y component in world coordinates

3

set the context to the values specified in the IAT and RAT arrays (as described above).

IAT

An integer array, of length 14, that has meaning only if IOPT equals 2 or 3.

RAT

A real array, of length 7, that has meaning only if IOPT equals 2 or 3.

If you want to save the state of the NCAR GKS primitive attribute settings, call NGSRAT with a first argument of 0. When the first argument is 0, the IAT and RAT arrays (second and third arguments) are ignored and the state values are stored internally to NGSRAT.

To restore the NCAR GKS primitive attribute values to what they were as of the last time NGSRAT was called with argument IOPT = 0, then call NGSRAT with first argument of 1. IAT and RAT are ignored in this case.

If you call NGSRAT with a fist argument of 2, then the current settings of the NCAR GKS primitive attribute values are returned in arrays IAT and RAT as per the description of IAT and RAT above.

If you call NGSRAT with a fist argument of 3, then arrays IAT and RAT must be supplied and the values therein will be used to set the NCAR GKS primitive attribute values as per the description of IAT and RAT above.

A common use of NGSRAT would be in conjunction with NGMFTC and NGREOP for saving the GKS state after writing to one metafile, writing to a new metafile, and then restoring the GKS state when reopening the initial metafile.

The following sequence:

```
        CALL NGSRAT(0,IAT,RAT)

          ... do some setting and drawing

        CALL NGSRAT(1,IAT,RAT)
```

would save the values of all NCAR GKS primitive attributes before the code that does setting and drawing and then restore the attribute values after the setting and drawing.

Do an "ncargex pgkex27" for a relevant example.

Common Pitfalls

Getting All of Your Output

A problem that often causes the loss of part (or all) of the last frame of a graphics session occurs because many of the NCAR Graphics LLUs do not automatically advance to a new picture when they are called. This is to allow you to write multiple graphics on a single picture and include graphics from multiple utilities. The user may forget to advance to a new picture at the end of an application as well. At the end of a job you must remember to issue a call to flush the graphics buffers (CALL SFLUSH), or a call to advance to the next picture (CALL FRAME) which will call SFLUSH. For details on SFLUSH or FRAME, consult the man pages.

Output Attributes Changing at the Wrong Time

For reasons of efficiency, SPPS maintains an internal buffer which accumulates polyline coordinates. This buffer is automatically flushed when it is full, or when FRAME is called, or it can be flushed by the user with a call to SFLUSH. If polyline attributes are changed between SPPS calls that produce lines, then failure to flush the polyline buffer in SPPS between the line-producing calls can cause unexpected results.

For example, if you set line color to pink, then call LINE to draw a line, then set line color to blue and call LINE to draw another line, and then call FRAME, you will get two blue lines unless you flush the polyline buffer after drawing the first line. This is because the order in which things are done is: put line color pink to the output stream; put out coordinates for the first line call in the SPPS buffer; put line color blue to the output stream; put coordinates for the second line in the SPPS buffer; put polyline elements to the output stream when the call to FRAME is encountered. If you are using direct GKS calls (such as GPL), then the buffer flush is not necessary because the buffer is a feature of SPPS and not a feature of GKS.

Getting the Font You Want

All of the Hershey fonts available through GSTXFP are available via the Plotchar package. PostScript source documentation for Plotchar is available. Plotchar also supports a number of filled, scalable, fonts. If you are using a PostScript output workstation, then the fonts of the PostScript interpreter used to draw the plots will be used.

The Dot Polymarker Cannot be Scaled

The GKS standard requires that marker type 1 always be displayed as the smallest possible dot. For this reason, the dot polymarker will not be affected by the marker size scale factor as set by GSMKSC. To get filled dots of any size and color use the NGDOTS subroutine.

Precedence of metafile names

If you use NGSETI (or escape function -1391) to set a value for 'ME' to provide a metafile name within your code, this name will be superceded by the name supplied as the setting of the environment variable NCARG_GKS_OUTPUT. Make sure that NCARG_GKS_OUTPUT is not set when specifying a metafile name in your code.

See the man pages for NGSETI, NGSETR, NGSETC, and ngmisc_params for setting values for NCAR GKS escape functions, such as a metafile name.

See ncargex to run example pgkex26 showing how to name metafiles within your code.

Positioning Character Strings Where You Want Them

When positioning characters it can often be convenient to use normalized device coordinates in the range of 0. to 1. in both dimensions, as opposed to an arbitrary set of units as defined by the user coordinate system. You should generate the desired text output, then return to the original set of user coordinates. The four steps involved are:

1. Call GQCNTN (ERRIND, TRNUM) to get TRNUM, the current normalization transformation number.
2. If TRNUM is not 0, invoke normalization transformation 0 with a call to GSELNT (0).
3. Write your text using Plotchar, or GTX.
4. Return to the original normalization transformation, TRNUM, with a call to GSELNT (TRNUM).

Links: GKS Index, GKS Home

Appendices

Appendix A. Codes Used to Generate Graphics in the NCAR GKS documentation modules

Most of the Fortran codes that were used to generate the figures in the GKS documentation modules (that is, those modules accessable from the GKS Homepage) are available via the ncargex command. The codes available from "ncargex" produce an NCGM by default, but "ncargex" has an option, "-Wnn", where "nn" is an NCAR GKS workstation type, that allows for PostScript or X11 output instead of NCGM output.

The graphics in the GKS documentation modules were originally created using workstation type 23. The output PostScript was then converted to GIF for inclusion in the on-line modules.

Table A1. Index of codes used to create graphics for the GKS modules

Figure	Description	ncargex example name
Figure 1	A typical GKS code layout	pgkex01
Figure 2	GKS output primitives	pgkex02
Figure 3	Using inquiry functions	pgkex03
Figure 4	GKS normalization transformation	pgkex04
Figure 5	Clipping	pgkex05
Figure 6	Polylines	pgkex06
Figure 7	Polymarkers	pgkex07
Figure 8	Font Coordinate system	pgkex08
Figure 9	Text extent rectangle	pgkex09
Figure 10	Text path	pgkex10
Figure 11	Text alignment	pgkex11
Figure 12	Text fonts available	pgkex12
Figure 13	Character spacing and expansion factor	pgkex13
Figure 14	Character height and up vector	pgkex14
Figure 15	Filled areas	pgkex15
Figure 16	Fill area algorithm	pgkex16
Figure 17	Fill area interior styles	pgkex17
Figure 18	Cell arrays	pgkex18
Figure 19	PostScript portrait mode	pgkex19
Figure 20	PostScript full page output	pgkex20
Figure 21	PostScript line styles	pgkex21
Figure 22	PostScript nominal linewidth	pgkex22
Figure 23	PostScript spacing between hatch lines	pgkex23
Figure 24	Log scaling with SPPS	pgkex24
Figure 25	Axis reversal with SPPS	pgkex25
(none)	Writing to multiple metafiles (no plot included)	pgkex26
(none)	Reopening NCGMs (no plot included)	pgkex27
Diagram 1	A typical GKS normalization transformation	(none)
Diagram 2	Mirror imaging and log scaling	(none)

Appendix B. NCAR GKS-0A Subroutine Subset

Table B1 shows the Fortran subroutines along with a summary of their function; Table B2 shows the equivalent C functions.

Table B1. NCAR GKS Fortran Calls in This Document

```
---------------------------------------------------------------------------
 Subroutine                                    Function
---------------------------------------------------------------------------
 GACWK (WKID) ........................ Activate workstation
 GCA (XP1,YP1,XP2,YP2,NXCLR,NYCLR,
      IXCLR,IYCLR,NXDO,NYDO,CLRIA) .... Cell array output
 GCLKS ............................... Close GKS
 GCLRWK (WKID,COFL) .................. Clear workstation
 GCLWK (WKID) ........................ Close workstation
 GDAWK (WKID) ........................ Deactivate workstation
 GESC (FCTID,LIDR,IDR,MXODR,LODR,ODR) . Escape
 GFA (N,X,Y) ......................... Fill area output
 GOPKS (ERFILE,BUFA) ................. Open GKS
 GOPWK (WKID,CONID,WKTYPE) ........... Open workstation
 GPL (N,X,Y) ......................... Polyline output
 GPM (N,X,Y) ......................... Polymarker output
 GQASF (LASF) ........................ Get aspect source flags
 GQCHH (CHH) ......................... Set Character Height
 GQCHSP (ERRIND,CHSP) ................ Get character spacing
 GQCHUP (ERRIND,CHUX,CHUY) ........... Get character up vector
 GQCHXP (ERRIND,CHXP) ................ Get character expansion factor
 GQCLIP (ERRIND,ICLIP,CLRECT) ........ Get clipping indicator
 GQCNTN (ERRIND,CTNR) ................ Id of current transformation
 GQCR (WKID,CI,TYPE,ERRIND,CR,CG,CB) .. Get color representation
 GQFACI (ERRIND,COLI) ................ Get fill area color index
 GQFAIS (ERRIND,INTS) ................ Get fill area interior style
 GQFASI (ERRIND,STYLI) ............... Get fill area style index
 GQLN (ERRIND,LTYPE) ................. Get linetype
 GQLWSC (ERRIND,LWIDTH) .............. Get linewidth
 GQMK (ERRIND,MTYPE) ................. Get marker type
 GQMKSC (ERRIND,MSZSF) ............... Get the marker size scale factor
 GQMNTN (ERRIND,MAXTRN) .............. Get no. of supported transformations
 GQNT (NTNR,ERRIND,WINDOW,VIEWPT) ..... Get window and viewport info
 GQOPS (OPSTA) ....................... Get open status of GKS
 GQPLCI (ERRIND,COLI) ................ Get polyline color index
 GQPMCI (ERRIND,COLI) ................ Get polymarker color index
 GQTXAL (ERRIND,TXALH,TXALV) ......... Get text alignment
 GQTXCI (ERRIND,COLI) ................ Get text color index
 GQTXFP (ERRIND,FONT, PREC) .......... Get text font and precision
 GQTXP (ERRIND,TXP) .................. Get text path
 GSASF (LASF) ........................ Set aspect source flags
 GSCHH (CHH) ......................... Set character height
 GSCHUP (CHUX,CHUY) .................. Set character up vector
 GSCHXP (CHXP) ....................... Set character expansion factor
 GSCHSP (CHSP) ....................... Set character spacing
 GSCLIP (ICLIP) ...................... Set clipping indicator
 GSCR (WKID,CI,CR,CG,CB).............. Set color representation
 GSELNT (TRNUM) ...................... Select normalization transformation
 GSFACI (COLI) ....................... Set fill area color index
 GSFAIS (INTS) ....................... Set fill area interior style
 GSFASI (STYLI) ...................... Set fill area style index
 GSLN (LTYPE) ........................ Set linetype
 GSLWSC (LWIDTH) ..................... Set linewidth
 GSMK (MTYPE) ........................ Set marker type
 GSMKSC (MSZSF) ...................... Set the marker size scale factor
 GSPLCI (COLI) ....................... Set polyline color index
 GSPMCI (COLI) ....................... Set polymarker color index
 GSTXAL (TXALH,TXALV) ................ Set text alignment
```

```
 GSTXCI (COLI) ........................ Set text color index
 GSTXFP (FONT, PREC) .................. Set text font and precision
 GSTXP (TXP) .......................... Set text path
 GSVP (TRNUM,XMIN,XMAX,YMIN,YMAX) ..... Set viewport
 GSWN (TRNUM,XMIN,XMAX,YMIN,YMAX) ..... Set window
 GTX (X,Y,STRING) ..................... Text output
 GUWK (WKID,REGFL) .................... Update workstation
-----------------------------------------------------------------------------
```

Table B2. NCAR GKS C Calls in This Document

```
-----------------------------------------------------------------------------
 C function (all C functions return void)
-----------------------------------------------------------------------------
 gactivate_ws(Gint ws_id)
 gcell_array(const Grect *rect, const Gpat_rep *colr_array)
 gclear_ws(Gint ws_id, Gctrl_flag ctrl_flag)
 gclose_gks(void)
 gclose_ws(Gint ws_id)
 gdeactivate_ws(Gint ws_id)
 gescape(Gint fid, const Gescape_in_data *in, Gstore *store,
         Gescape_out_data **out)
 gfill_area(const Gpoint_list *point_list)
 ginq_asfs(Gint *err_ind, Gasfs *list_asf)
 ginq_char_expan(Gint *err_ind, Gdouble *char_expan)
 ginq_char_ht(Gint *err_ind, Gdouble *char_ht)
 ginq_char_space(Gint *err_ind, Gdouble *char_space)
 ginq_char_up_vec(Gint *err_ind, Gvec *char_up_vec)
 ginq_clip(Gint *err_ind, Gclip *clip_ind_rect)
 ginq_colr_rep(Gint ws_id, Gint ind, Ginq_type type, Gint *err_ind,
               Gcolr_rep *colr_rep)
 ginq_cur_norm_tran_num(Gint *err_ind, Gint *tran_num)
 ginq_fill_colr_ind(Gint *err_ind, Gint *fill_colr_ind)
 gset_fill_int_style(Gfill_int_style fill_int_style)
 gset_fill_style_ind(Gint fill_style_ind)
 ginq_line_colr_ind(Gint *err_ind, Gint *line_colr_ind)
 ginq_linetype(Gint *err_ind, Gint *linetype)
 ginq_linewidth(Gint *err_ind, Gdouble *linewidth)
 ginq_marker_colr_ind(Gint *err_ind, Gint *marker_colr_ind)
 ginq_marker_size(Gint *err_ind, Gdouble *marker_size)
 ginq_marker_type(Gint *err_ind, Gint *marker_type)
 ginq_max_norm_tran_num(Gint *err_ind, Gint *max_num)
 ginq_norm_tran(Gint num, Gint *err_ind, Gtran *norm_tran)
 ginq_op_st(Gop_st *op_st)
 ginq_text_align(Gint *err_ind, Gtext_align *text_align)
 ginq_text_color_ind(Gint *err_ind, Gint *text_colr_ind)
 ginq_text_font_prec(Gint *err_ind, Gtext_font_prec *prec)
 ginq_text_path(Gint *err_ind, Gtext_path *text_path)
 gopen_gks(const char *err_file,  size_t mem_unit)
 gopen_ws(Gint ws_id, const char *conn_id, Gint ws_type)
 gpolyline(const Gpoint_list *point_list)
 gpolymarker(const Gpoint_list *point_list)
 gsel_norm_tran(Gint tran_num)
 gset_asfs(const Gasfs *list_asf)
 gset_char_expan(Gdouble char_expan)
 gset_char_ht(Gdouble char_ht)
 gset_char_space(Gdouble char_space)
 gset_char_up_vec(const Gvec *char_up_vec)
 gset_clip_ind(Gclip_ind clip_ind)
 gset_colr_rep(Gint ws_id, Gint ind, const Gcolr_rep *crep)
```

```
gset_fill_colr_ind(Gint fill_colr_ind)
gset_fill_int_style(Gfill_int_style fill_int_style)
gset_fill_style_ind(Gint fill_style_ind)
gset_line_colr_ind(Gint line_colr_ind)
gset_linetype(Gint linetype)
gset_linewidth(Gdouble linewidth)
gset_marker_colr_ind(Gint marker_colr_ind)
gset_marker_size(Gdouble marker_size)
gset_marker_type(Gint marker_type)
gset_text_align(const Gtext_align *text_align)
gset_text_colr_ind(Gint text_colr_ind)
gset_text_font_prec(const Gtext_font_prec *text_font_prec)
gset_text_path(Gtext_path text_path)
gset_vp(Gint tran_num, const Glimit *vp_limits)
gset_win(Gint tran_num, const Glimit *win_limits)
gtext(const Gpoint *text_pos, const char *char_string)
gupd_ws(Gint ws_id, Gupd_regen_flag upd_regen_flag)
---------------------------------------------------------------------------
```

Appendix C. C Type Definitions Defined in gks.h.

Environmental type definitions.

Gdouble - double precision

```
typedef  double  Gdouble;
```

Gfloat - floating point number

```
typedef  float  Gfloat;
```

Gint - integer

```
typedef  int  Gint;
```

Other type definitions.

Gasf - aspect source flag

```
typedef enum {
    GASF_BUNDLED,
    GASF_INDIV
} Gasf;
```

Gasfs - aspect source flags

```
typedef struct {
    Gasf linetype;
    Gasf linewidth;
    Gasf line_colr_ind;
    Gasf marker_type;
    Gasf marker_size;
    Gasf marker_colr_ind;
    Gasf text_font_prec;
    Gasf char_expan;
```

```
    Gasf char_space;
    Gasf text_colr_ind;
    Gasf fill_int_style;
    Gasf fill_style_ind;
    Gasf fill_colr_ind;
} Gasfs;
```

Gclip - clipping

```
typedef struct {
    Gclip_ind clip_ind;   /* clipping indicator */
    Glimit    clip_rect;  /* clipping rectangle */
} Gclip;
```

Gclip_ind - clipping indicator

```
typedef enum {
    GIND_NO_CLIP,
    GIND_CLIP
} Gclip_ind;
```

Gcolr_rep - color representation

```
typedef union {
    Grgb   rgb;  /* Red Green Blue colour specification  */
} Gcolr_rep;
```

Gctrl_flag - control flag

```
typedef enum {
    GFLAG_COND,
    GFLAG_ALWAYS
} Gctrl_flag;
```

Gdata - data

```
typedef struct {
    size_t size;   /* size of data    */
    void   *data;  /* pointer to data */
} Gdata;
```

Gescape_in_data - escape input data record

```
typedef union {
    Gdata escape_u1390;
    Gdata escape_u1391;
    Gdata escape_u1392;
    Gdata escape_u1393;
    Gdata escape_u1394;
    Gdata escape_u1395;
    Gdata escape_u1396;
    Gdata escape_u1397;
    Gdata escape_u1398;
    Gdata escape_u1399;
    Gdata escape_u1400;
    Gdata escape_u1401;
    Gdata escape_u1510;
    Gdata escape_u1511;
```

```
      Gdata escape_u1512;
      Gdata escape_u1513;
      Gdata escape_u1514;
      Gdata escape_u1515;
      Gdata escape_u1516;
      Gdata escape_u1517;
      Gdata escape_u1518;
      Gdata escape_u1519;
      Gdata escape_u1520;
      Gdata escape_u1521;
      Gdata escape_u1522;
  } Gescape_in_data;
```

Gescape_out_data - escape output data record

```
  typedef union {
         Gdata escape_u1390;
  } Gescape_out_data;
```

Gfill_int_style - fill area interior style

```
  typedef enum {
      GSTYLE_HOLLOW,
      GSTYLE_SOLID,
      GSTYLE_PAT,
      GSTYLE_HATCH
  } Gfill_int_style;
```

Ghor_text_align - horizontal text alignment

```
  typedef enum {
      GHOR_NORM,
      GHOR_LEFT,
      GHOR_CTR,
      GHOR_RIGHT
  } Ghor_text_align;
```

Ginq_type - inquire type

```
  typedef enum {
      GINQ_SET,
      GINQ_REALIZED
  } Ginq_type;
```

Glimit - limit

```
  typedef struct {
      Gfloat x_min;  /* x min */
      Gfloat x_max;  /* x max */
      Gfloat y_min;  /* y min */
      Gfloat y_max;  /* y min */
  } Glimit;
```

Gop_st - operating state

```
  typedef enum {
      GST_GKCL,
      GST_GKOP,
```

```
    GST_WSOP,
    GST_WSAC,
    GST_SGOP
} Gop_st;
```

Gpat_rep - pattern representation

```
typedef struct {
    Gint_size dims;         /* pattern's dimensions */
    Gint      *colr_array;  /* colour array         */
} Gpat_rep;
```

Gpoint - point

```
typedef struct {
    Gfloat x;       /* x coordinate */
    Gfloat y;       /* y coordinate */
} Gpoint;
```

Gpoint_list - point list

```
typedef struct {
    Gint   num_points;  /* num. of points in the list */
    Gpoint *points;     /* list of points             */
} Gpoint_list;
```

Grect - rectangle

```
typedef struct {
    Gpoint p;        /* point p */
    Gpoint q;        /* point q */
} Grect;
```

Grgb - Red Green Blue [color specification]

```
typedef struct {
    Gfloat   red;   /* red intensity   */
    Gfloat green;   /* green intensity */
    Gfloat  blue;   /* blue intensity  */
} Grgb;
```

Gstore - store

```
typedef void       *Gstore;
```

Gtext_align - text alignment

```
typedef struct {
    Ghor_text_align   hor;   /* horizontal component */
    Gvert_text_align  vert;  /* vertical component   */
} Gtext_align;
```

Gtext_font_prec - text font [and] precision

```
typedef struct {
    Gint         font;  /* text font      */
    Gtext_prec   prec;  /* text precision */
```

```
} Gtext_font_prec;
```

Gtext_path - text path

```
typedef enum {
    GPATH_RIGHT,
    GPATH_LEFT,
    GPATH_UP,
    GPATH_DOWN
} Gtext_path;
```

Gtext_prec - text precision

```
typedef enum {
    GPREC_STRING,
    GPREC_CHAR,
    GPREC_STROKE
} Gtext_prec;
```

Gtran - transformation

```
typedef struct {
    Glimit win;          /* window   */
    Glimit  vp;          /* viewport */
} Gtran;
```

Gupd_regen_flag - update regeneration flag

```
typedef enum {
    GFLAG_POSTPONE,
    GFLAG_PERFORM
} Gupd_regen_flag;
```

Gvec - vector

```
typedef struct {
    Gfloat delta_x;  /* x coordinate */
    Gfloat delta_y;  /* y coordinate */
} Gvec;
```

Gvert_text_align - vertical text alignment

```
typedef enum {
    GVERT_NORM,
    GVERT_TOP,
    GVERT_CAP,
    GVERT_HALF,
    GVERT_BASE,
    GVERT_BOTTOM
} Gvert_text_align;
```

Appendix D. NCAR GKS-0A Error Messages

NCAR GKS-0A Error List

```
----------------------------------------------------------------------------
```

Error No.	Message
1	GKS NOT IN PROPER STATE: GKS SHALL BE IN STATE GKCL
2	GKS NOT IN PROPER STATE: GKS SHALL BE IN STATE GKOP
3	GKS NOT IN PROPER STATE: GKS SHALL BE IN STATE WSAC
4	GKS NOT IN PROPER STATE: GKS SHALL BE IN STATE SGOP
5	GKS NOT IN PROPER STATE: GKS SHALL BE EITHER IN THE STATE WSAC OR IN THE STATE SGOP
6	GKS NOT IN PROPER STATE: GKS SHALL BE EITHER IN THE STATE WSOP OR IN THE STATE WSAC
7	GKS NOT IN PROPER STATE: GKS SHALL BE IN ONE OF THE STATES WSOP, WSAC, OR SGOP
8	GKS NOT IN PROPER STATE: GKS SHALL BE IN ONE OF THE STATES GKOP, WSOP, WSAC, OR SGOP
20	SPECIFIED WORKSTATION IDENTIFIER IS INVALID
21	SPECIFIED CONNECTION IDENTIFIER IS INVALID
22	SPECIFIED WORKSTATION TYPE IS INVALID
23	SPECIFIED WORKSTATION TYPE DOES NOT EXIST
24	SPECIFIED WORKSTATION IS OPEN
25	SPECIFIED WORKSTATION IS NOT OPEN
26	SPECIFIED WORKSTATION CANNOT BE OPENED
27	WORKSTATION INDEPENDENT SEGMENT STORAGE IS NOT OPEN
28	WORKSTATION INDEPENDENT SEGMENT STORAGE IS ALREADY OPEN
29	SPECIFIED WORKSTATION IS ACTIVE
30	SPECIFIED WORKSTATION IS NOT ACTIVE
32	SPECIFIED WORKSTATION IS NOT OF CATEGORY MO
33	SPECIFIED WORKSTATION IS OF CATEGORY MI
34	SPECIFIED WORKSTATION IS NOT OF CATEGORY MI
35	SPECIFIED WORKSTATION IS OF CATEGORY INPUT
36	SPECIFIED WORKSTATION IS WORKSTATION INDEPENDENT SEGMENT STORAGE
38	SPECIFIED WORKSTATION IS NEITHER OF CATEGORY INPUT NOR OF CATEGORY OUTIN
50	TRANSFORMATION NUMBER IS INVALID
51	RECTANGLE DEFINITION IS INVALID
52	VIEWPORT IS NOT WITHIN THE NORMALIZED DEVICE COORDINATE UNIT SQUARE
53	WORKSTATION WINDOW IS NOT WITHIN THE NORMALIZED DEVICE COORDINATE UNIT SQUARE
54	WORKSTATION VIEWPORT IS NOT WITHIN THE DISPLAY SPACE
60	POLYLINE INDEX IS INVALID
63	LINETYPE IS LESS THAN OR EQUAL TO ZERO
65	LINEWIDTH SCALE FACTOR LESS THAN ZERO
66	POLYMARKER INDEX IS INVALID
69	MARKER TYPE IS LESS THAN OR EQUAL TO ZERO
72	TEXT INDEX IS INVALID
75	TEXT FONT IS EQUAL TO ZERO
77	CHARACTER EXPANSION FACTOR IS LESS THAN OR EQUAL TO ZERO
78	CHARACTER HEIGHT IS LESS THAN OR EQUAL TO ZERO
79	LENGTH OF CHARACTER UP VECTOR IS ZERO
80	FILL AREA INDEX IS INVALID
84	STYLE (PATTERN OR HATCH) INDEX IS LESS THAN OR EQUAL TO ZERO
87	PATTERN SIZE VALUE IS NOT POSITIVE
91	DIMENSIONS OF COLOR ARRAY ARE INVALID
92	COLOR INDEX IS LESS THAN ZERO
93	COLOR INDEX IS INVALID
96	COLOR IS OUTSIDE RANGE ZERO TO ONE INCLUSIVE
100	NUMBER OF POINTS IS INVALID
101	INVALID CODE IN STRING
102	GENERALIZED DRAWING PRIMITIVE IDENTIFIER IS INVALID
103	CONTENT OF GENERALIZED DRAWING PRIMITIVE DATA RECORD IS INVALID
104	AT LEAST ONE ACTIVE WORKSTATION IS NOT ABLE TO GENERATE THE

	SPECIFIED GENERALIZED DRAWING PRIMITIVE
120	SPECIFIED SEGMENT NAME IS INVALID
121	SPECIFIED SEGMENT NAME IS ALREADY IN USE
122	SPECIFIED SEGMENT DOES NOT EXIST
124	SPECIFIED SEGMENT DOES NOT EXIST ON WORKSTATION INDEPENDENT SEGMENT STORAGE
160	ITEM TYPE IS NOT ALLOWED FOR USER ITEMS
161	ITEM LENGTH IS INVALID
162	NO ITEM IS LEFT IN GKS METAFILE INPUT
163	METAFILE ITEM IS INVALID
165	CONTENT OF ITEM DATA RECORD IS INVALID FOR THE SPECIFIED ITEM TYPE
166	MAXIMUM ITEM DATA RECORD LENGTH IS INVALID
167	USER ITEM CANNOT BE INTERPRETED
168	SPECIFIED FUNCTION IS NOT SUPPORTED IN THIS LEVEL OF GKS
180	SPECIFIED FUNCTION IS NOT SUPPORTED
182	CONTENTS OF ESCAPE DATA RECORD ARE INVALID
300	STORAGE OVERFLOW HAS OCCURRED IN GKS
302	INPUT/OUTPUT ERROR HAS OCCURRED WHILE READING
303	INPUT/OUTPUT ERROR HAS OCCURRED WHILE WRITING
304	INPUT/OUTPUT ERROR HAS OCCURRED WHILE SENDING DATA TO A WORKSTATION
305	INPUT/OUTPUT ERROR HAS OCCURRED WHILE RECEIVING DATA FROM A WORKSTATION
306	INPUT/OUTPUT ERROR HAS OCCURRED DURING PROGRAM LIBRARY MANAGEMENT
307	INPUT/OUTPUT ERROR HAS OCCURRED WHILE READING WORKSTATION DESCRIPTION TABLE
308	ARITHMETIC ERROR HAS OCCURRED
-100	UNKNOWN ERROR CODE
-101	NO ADDITIONAL WORKSTATIONS MAY BE ACTIVATED
-102	GKS SYSTEM ERROR--IMPROPER CONTINUATION SEQUENCE
-103	THE NCAR GKS IMPLEMENTATION REQUIRES THAT WISS BE ACTIVE BEFORE A SEGMENT IS CREATED
-105	ERROR OPENING DISK FILE
-106	LOGICAL UNIT NUMBER FOR SEGMENT STORAGE CANNOT BE THE SAME AS THAT FOR METAFILE OUTPUT
-107	MAXIMUM NUMBER OF ERROR MESSAGES EXCEEDED
-108	ILLEGAL PLACEMENT OF A CALL TO ASSIGN A PICTURE NAME
-109	THIS FUNCTION NOT YET IMPLEMENTED FOR WORKSTATIONS OF CATEGORY OUTPUT
-110	CHARACTER STRING TOO LARGE
-111	SYSTEM ERROR
-112	ONLY ONE METAFILE WORKSTATION CAN BE OPEN AT A TIME
-200	X driver error: memory allocation in processing a character from a fontcap
-201	X driver error: error opening fontcap
-202	X driver error: error reading fontcap
-203	X driver warning: invalid font index, using the default font
-204	X driver error: memory allocation error in creating device table
-205	X driver error: too many open devices
-206	X driver error: error in internal memory management
-207	X driver error: error in allocating memory for device dependent table
-208	X driver error: DISPLAY environment variable not set
-209	X driver error: error opening display
-210	X driver error: error getting window attributes
-211	X driver error: error creating pixmap
-212	X driver error: cell array has zero width or height
-213	X driver error: memory allocation error in processing cell array
-214	X driver error: error creating X image
-215	X driver error: memory allocation error in creating X image
-216	X driver error: pixel size must be byte multiple
-217	X driver error: error in retrieving fontcap name

```
-218   X driver error: invalid index into GKS workstation identifiers
-219   X driver error: color index exceeds maximum allowed
-220   X driver warning: color allocated is different from the color requested
-221   X driver error: unable to change color model
-300   PostScript error: Encapsulated PostScript can have only one page
-301   PostScript error: error in allocating memory for device dependent
         table
-302   PostScript error: error opening output file
-303   PostScript warning: requested character not available, asterisk
         plotted, use Plotchar
2000   ENUMERATION TYPE OUT OF RANGE
2001   OUTPUT PARAMETER SIZE INSUFFICIENT
2002   LIST OR SET ELEMENT NOT AVAILABLE
2003   INVALID DATA RECORD
```

Appendix E. Supplemental Documentation

Learning More about GKS

- *Computer Graphics---Graphical Kernel System (GKS) Function Description*, ANSI X3.124, 1985.
- *Computer Graphics Programming---GKS---The Graphics Standard*, G. Enderle et al, Springer-Verlag, 1987.
- *GKS Primer*, L. McKay, NOVA Graphics International, 1984.
- *Introduction to the Graphical Kernel System (GKS),* F.R.A. Hopgood et al, Academic Press, 1986.

Links: GKS Index, GKS Home

GKS Index

The alphabetic characters that follow entries in this index refer to multiple citations. For example, if *Entry* is followed by "A,B", then "A" and "B" are links to two references to *Entry* in the GKS documentation modules.

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

NCARG GKS 0A Credits

Acknowledgments

Author: Fred Clare

Editor: Tim Scheitlin

Technical Reviewer: Tom Parker

Production: Tim Scheitlin

Copyright

Trademarks

Published by:

National Center for Atmospheric Research, Scientific Computing Division, P.O. Box 3000, Boulder, CO 80307-3000. The National Center for Atmospheric Research is operated by the University Corporation for Atmospheric Research and is sponsored by the National Science Foundation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.