SLAC–390
UC–405
(M)

GKS–EZ
PROGRAMMING MANUAL FOR FORTRAN–77*

Robert C. Beach

Computation Research Group

Stanford Linear Accelerator Center

Stanford University

Stanford, CA 94309

January 1992

Prepared for the Department of Energy

under contract number DE–AC03–76SF00515.

Printed in the United States of America. Available from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

⋆ Manual.

Contents

Chapter 1

Introduction

A standard has now been adopted for subroutine packages that drive graphic devices. It is known as the Graphical Kernel System (GKS), and many commercial implementations of it are available. Unfortunately, it is a difficult system to learn, and certain functions that are important for scientific use are not provided. Although GKS can be used to achieve portability of graphic applications between graphic devices, computers, and operating systems, it can also be misused in this respect. In addition, it introduces the very real problem of portability between the various implementations of GKS.

This document describes a set of FORTRAN-77 subroutines that may be used to control a wide variety of graphic devices and overcome most of these problems. Some of these subroutines are from GKS itself, while others are higher-level subroutines that call GKS subroutines. These subroutines are collectively known as GKS-EZ. The purpose is to supply someone who is not a specialist in computer graphic with a flexible, robust, and easy to learn graphics system. Users of GKS-EZ should not have much need for a full GKS manual; this document will supply all of the information to use GKS-EZ except for a few items. These missing items include the numeric identification of the supported graphic devices and the procedure for linking the GKS subroutines into an executable module.

However, it must be emphasized that the design goals of GKS and GKS-EZ are different. The emphasis in GKS is to provide the programmer with all of the tools necessary to drive a very wide variety of graphic devices. GKS-EZ, on the other hand, tries to provide a means of writing programs which will run on a wide variety of graphic devices. These two goals are *not* the same.

GKS-EZ is fully compatible with the standardization efforts of the American National Standards Institute (ANSI). In particular, it is compatible with the ASCII character set as described in *American National Standard for Information Systems: Coded Character Sets, 7-bit American National Standard Code for Information Interchange (7-bit ASCII)* [ANS86], FORTRAN-77 as described in *American National Standard: Programming Language FORTRAN* [ANS78], and GKS itself as described in *American National Standard for Information Systems: Computer Graphics - Graphical Kernel System (GKS) Functional Description* [ANS85a] and *American National Standard for Information Systems: Computer Graphics - Graphical Kernel System (GKS) FORTRAN Binding* [ANS85b].

Many concepts will have to be defined in the following chapters. When a concept is first encountered, it will be given in *italics*. The information around the italicized word or phrase may be taken as its definition.

1.1. The Graphical Kernel System (GKS)

GKS has now been adopted by both ANSI and the International Standards Organization (ISO). It is therefore an American and an international standard. The standardization effort includes the standardization of the calling sequences for FORTRAN and other languages. This section will provide a basic introduction to most of the concepts used in GKS.

GKS contains three separate two-dimensional Cartesian coordinate systems. The coordinates within all of these coordinate systems are specified by real values. The three coordinate systems are:

1. The World Coordinate System (WC): This is the system in which the user normally specifies the positions of graphic primitives. Its extent is potentially infinite in x and y.
2. Normalized Device Coordinates (NDC): An intermediary coordinate system. It is a uniform coordinate system for all graphic devices. Its extent is essentially limited by 0.0 to 1.0 in x and y.
3. Device Coordinates (DC): The coordinate system of the actual graphic device. Its extent is from 0.0 to `XDCMAX` in x and from 0.0 to `YDCMAX` in y. `XDCMAX` and `YDCMAX` are specified in meters or other units and are device-dependent; their values may be obtained at execution time.

Transformations between these coordinate systems are specified by giving a rectangle with sides parallel to the coordinate axes in each of two coordinate systems. The transformation is then defined as a linear mapping from one rectangle to the other. When the transformation is thought of as going from a source coordinate system to a target coordinate system, then the rectangle in the source coordinate system is called a *window* while the rectangle in the target coordinate system is called a *viewport*. GKS allows the user to control two transformations:

1. The Normalization Transformation: This transforms from the world coordinate system to normalized device coordinates. GKS allows multiple normalization transforms to be available at one time. The aspect ratios of the window and viewport are unconstrained for this transformation.
2. The Workstation Transformation: This transforms from normalized device coordinates to device coordinates. The transformation may be different for each graphic device. For this transformation, the aspect ratio of the window and viewport must be the same.

GKS supplies a default normalization transformation and workstation transformation that results in the square with x and y running from 0.0 to 1.0 in world coordinates mapping into a maximal square area in device coordinates.

The graphic primitives in GKS are of six types:

1. Polylines: A polyline consists of a sequence of concatenated line segments.
2. Polymarkers: A polymarker is a group of markers. Each marker can be a single point or a more elaborate plotting symbol.
3. Text: A text primitive consists of a single string of characters.
4. Fill Area: This primitive consists of a polygonal area that may be filled with solid colors, patterns, or cross-hatching.

5. Cell Array: A rectangular array of cells whose color can be individually specified.
6. Generalized Drawing Primitive: An escape function that allows a programmer to exploit special features of certain graphic devices.

The positions of all of these primitives are specified by giving the x and y coordinates of points in the world coordinate system. The attributes of these primitives, such as color, size, and line type, may also be specified. Two methods of specifying attributes, called ***individual*** and ***bundled***, are supplied in GKS. When individual attributes are used, they apply to all graphic devices in use. Bundled attributes are user defined collections of attributes which may be assigned to individual graphic devices.

Under GKS, a picture may optionally consist of a number of parts called *segments*. Each segment can consist of an arbitrary number of graphic primitives and is assigned a numeric identification. Segments are the smallest objects that may be manipulated by GKS. For example, segments may be highlighted or deleted. GKS also allows segments to be rotated, translated, and scaled under certain conditions.

Another basic concept of GKS is the *workstation*. A workstation may be a plotter, a simple graphic terminal, or a more flexible graphic device. GKS supports six types of workstations:

1. OUTPUT: Output only. An example is a simple plotter.
2. INPUT: Input only. The best example is probably a non-graphic terminal.
3. OUTIN: Output and Input. Typically, this is an interactive graphic terminal.
4. WISS: Workstation Independent Segment Storage. This allows a programmer to temporarily store and manipulate segments. The segments are only available from within a single program.
5. MO: Metafile Output. This allows a job to save pictures in a file for retrieval by another program.
6. MI: Metafile Input. Pictures prepared for an MO workstation may be retrieved.

A workstation operator may specify information to a program by using graphic input devices. Graphic input devices are grouped into six classes in GKS:

1. Locator: This type of device returns the x and y coordinates of a point in world coordinates to the program. A mouse or other pointing device can function as a locator control unit.
2. Stroke: This device returns a sequence of points. A mouse can also act as a stroke control unit.
3. Valuator: A real number is returned. A rotary dial can act as a valuator control unit.
4. Choice: A non-negative integer is returned that represents a selection from a number of choices. A set of push buttons can be a choice control unit.
5. Pick: The identification of a segment and the identification of a graphic primitive within the segment can be returned. A light pen is an example of a pick control unit.

6. String: A character string is returned to the program. A keyboard is an example of a string control unit.

GKS then allows these six types of input devices to be used in three modes:

1. Request: GKS waits until the input is supplied by the workstation operator or until a break action is performed.
2. Sample: The current value of the control unit is returned immediately to the program.
3. Event: The data from the control unit is put on an input queue. The application program can then remove the oldest event from the queue and examine it.

A workstation will be in one of three possible states at any time. These three states are:

1. The workstation is closed. In this state, absolutely no output to the workstation or input from the workstation may be performed.
2. The workstation is open but not active. In this state, input may be obtained from the workstation but no graphic output can be written to it.
3. The workstation is active. In this state, input may also be obtained from the workstation. In addition, any graphic output that is generated will be sent to the workstation.

A given implementation of GKS does not have to perform all of the functions defined in the standards document. GKS implementations are classified into various levels, depending on how much of GKS they actually support. There are four levels of output and three levels of input that are explicitly defined in the ANSI version of GKS (the ISO version only has three levels of output; the "m" level is not defined). Each level has all of the functions of the preceding levels. The levels for output are:

m. Minimal output. All primitives except cell array are available, but only a very limited set of attributes may be assigned to them.
0. All primitives and attributes are available. However, segments are not yet available.
1. At this level, segments and bundled attributes are available.
2. Workstation independent segment storage becomes available at this level.

The GKS levels for input are:

a. No input functions are available.
b. Only the request input mode is available.
c. All input modes are available.

Therefore, there are a total of twelve possible levels for a GKS implementation. The GKS level is a two-character identification formed from the output and input level specifications. Thus, the most primitive level is "Level ma" while a full GKS implementation is "Level 2c."

1.2. The Rationale for GKS-EZ

GKS is now an American and International standard and cannot be ignored. Since the FORTRAN-77 binding has also been standardized, the transportability

of application programs from one computer or operating system could be relatively straightforward. Nevertheless, there are substantial problems in using GKS. Some of these problems are:

1. GKS consists of an extremely large number of subroutines. The FORTRAN-77 binding specifies 212 subroutines for Level 2c. Even the lowest level, Level ma, consists of 56 subroutines and that level is only marginally adequate for the simplest non-interactive devices. Learning that many subroutines, and learning which of them are really important, will clearly take time. GKS, therefore, appears unsuitable for anyone who does not intend to make a career of computer graphics.
2. GKS requires that many attributes and other flags be specified by integer values; for example, a solid line is selected by a value of 1, while a dashed line is selected by a 2. Programs that use integer constants will quickly become unreadable. The FORTRAN-77 binding does define *enumeration types* which give symbolic names to these integer values using the FORTRAN-77 **PARAMETER** statement. However, experience has shown that most programmers will ignore the enumeration types unless they are coerced into using them. It is initially quicker and easier not to use the enumeration types, and most programmers will take the easy way.
3. Full GKS provides more than one way to accomplish a particular function. This redundancy can be confusing and distracting to beginning users and to users who are not computer graphics specialists.
4. Portability across different implementations of GKS is possible, but there are many potential problems. For example, the number of line types, marker types, or attribute bundles can vary between implementations. Any use of such facilities beyond those defined in the standard can limit portability.
5. GKS has very poor support for producing textual material for scientific applications. These applications require, at a minimum, the free intermixing of Roman and Greek letters, and superscripting and subscripting. The only characters defined in the GKS standards document are a single font of characters containing the ASCII character set. All implementations of GKS will probably have additional fonts containing Greek letters and other symbols. However, the fonts containing them are not standardized so applications using them may not be transportable across GKS implementations. In addition, the mixing of different fonts in a single line of text, or forming superscripts or subscripts, can lead to very tedious programming.
6. GKS allows the writing of very device-dependent programs. Some of these device dependencies are obvious while others are not, especially to the non-specialist.

It is for these and other reasons that GKS-EZ was produced.

GKS-EZ is a small collection of subroutines that can be used to produce pictures on, and interact with, a variety of graphic devices. It is based on GKS and consists of the following:

1. Native GKS subroutines.

2. Subroutines that call groups of GKS subroutines, thereby providing simplified higher-level control.
3. Subroutines that call GKS subroutines and extend the capabilities of GKS itself.

Using GKS-EZ instead of GKS should not limit the user in any way. In particular, multiple graphic devices may be used in a single program, both non-interactive and fully interactive devices may be used, and essentially all transportable graphic primitives and attributes are supported. Some of the advantages of using GKS-EZ over plain GKS are:

1. GKS-EZ provides the beginning GKS user, and the GKS user who is not a computer graphics specialist, with a simplified set of subroutines that are fully compatible with GKS. The recommended GKS-EZ subroutines are only 31 in number.
2. A user of GKS-EZ will find that many applications can be done more easily and with far fewer subroutine calls than can be done with GKS itself. At the same time, the user of GKS-EZ is not precluded from using any of the native GKS subroutines. A certain amount of knowledge about the actions taken by the GKS-EZ subroutines may be necessary, but the source for these subroutines is readily available. Because of the structure of GKS-EZ subroutines, most application programs should be easier to read than programs using GKS directly.
3. A very extensive character generator containing the upper and lower case Roman, Greek, Cyrillic, and Hebrew alphabets, a wide variety of special characters, and a flexible superscripting and subscripting ability is provided. These characters can be produced in three different fonts. Since these characters are drawn with polylines and fill areas, they should appear essentially the same on any supported graphic device running under any GKS implementation.
3. Since GKS-EZ is built on top of GKS, it retains all of the transportability of GKS applications. Also, its carefully chosen subset of GKS subroutines greatly enhances the implementation-independence and device-independence of application programs.
5. Finally, the time spent in learning GKS-EZ will not be wasted if a user has to use GKS directly at a later time. In that case, the user will have a solid knowledge of part of GKS, and all of the nomenclature and conventions of GKS-EZ are the same as those of GKS itself.

1.3. The Availability of GKS-EZ

The subroutines described in this document are available on the IBM mainframe computers running at the Stanford Linear Accelerator Center. These computers are running under the VM/XA operating system. Executable versions of the subroutines are contained in the file

```
GKSEZ TXTLIB U.
```

They may therefore be used by anyone at this installation who uses the proper `TXTLIB` statement.

The source code is also available for those people who have to use the subroutines on another computer. The file

```
GKSEZ FORTRAN U
```

contains all of the subroutines described here.

Since the GKS-EZ subroutines are written in something very close to strict FORTRAN-77, they themselves should be transportable to any computer with a FORTRAN-77 compiler and a GKS system. The only non-standard construction in the source code is the use of `INTEGER*2` arrays to store the definition of the character sets. These declarations can easily be changed to `INTEGER`; the only requirement is that the arrays can contain integers of up to 32767.

There are two control values, `GZLWSF` and `GZMSSF`, that may have to be modified in some versions to GKS-EZ. The first control value is a line width scale factor multiplier and the second is a marker size scale factor multiplier. The problem is that the default line width and marker size are not specified in the standard. The result is that different implementation may produce very different line widths or marker sizes for the default. These control values may be changed from their initial value of 1.0 to make the default line width and marker sizes compatible. Unfortunately, this can mean that slightly different versions of GKS-EZ will have to be maintained for different GKS implementations, or even for different workstations within an implementation.

To use all of the functions in GKS-EZ, a GKS implementation of Level 1b is required. The GKS implementation must supply at least eight color indices, zero through seven, for each supported workstation.

Chapter 2

A Detailed Description of the GKS-EZ Subroutines

This chapter describes most of the subroutines available in GKS-EZ. It is meant to be a complete and self-contained description of GKS-EZ. Although not all functions of GKS are described, the subset of functions that are described here should be adequate for the vast majority of graphic application programs.

When a subroutine parameter with discrete values is described, its numerical value as well as its GKS enumeration type is given.

Error handling in GKS-EZ is quite simple. If one of the GKS-EZ subroutine additions detects an error, it prints a message on FORTRAN unit 6 and either tries to continue or, in rare cases, terminates. If a GKS subroutine detects an error, the normal GKS error processing will occur.

Each section in this chapter terminates with a description of the GKS implementation of the subroutines in that section. This information is usually not necessary for users of GKS-EZ but should be helpful when an application must use GKS subroutines that are not part of GKS-EZ.

The subroutines that belong to GKS-EZ, and not to GKS itself, are easily recognized; their names all start with "GZ." GKS itself does not have any subroutine names or enumeration types that begin with these letters.

2.1. Control Functions

The first four subroutines in this section are used to control the state of the workstation. The first two subroutines are used to open and close a workstation and are always necessary in an application program. The second two subroutines are used to activate and deactivate a workstation and are only necessary when a program is controlling more than one workstation. Even then, these two subroutines may not be necessary. The fifth subroutine is used to clear the display surface on a workstation and is always important.

2.1.1. Subroutine GZOPWK: Open Workstation

This subroutine may be used to open a workstation. If the workstation has output capability, it will be left in the active state. For most application programs, this subroutine will be called exactly once at the very beginning of the program. However, it may be called many times if, for example, the application program is controlling many workstations.

The calling sequence is:

```
CALL GZOPWK(WKID,CONID,WTYPE)
```

The input parameters are:

- `WKID` An integer giving the workstation identifier. This user selected value is the identification by which this workstation is referred to in all other subroutines. If more than one workstation is to be opened, each one must have a unique identifier.
- `CONID` An integer giving the connection identifier. This value is dependent on the GKS implementation being used.
- `WTYPE` An integer giving the workstation type. This value is dependent on the GKS implementation being used.

2.1.2. Subroutine GZCLWK: Close Workstation

This subroutine may be used to close a workstation. After a workstation has been closed, no more use can be made of it until it is reopened. For most application programs, this subroutine will be called exactly once at the very end of the program.

The calling sequence is:

```
CALL GZCLWK(WKID)
```

The input parameter is:

- `WKID` An integer giving the workstation identifier.

2.1.3. Subroutine GACWK: Activate Workstation

This subroutine may be used to activate a workstation. When a workstation is active, output primitives and segments will be written to it.

The calling sequence is:

```
CALL GACWK(WKID)
```

The input parameter is:

- `WKID` An integer giving the workstation identifier.

2.1.4. Subroutine GDAWK: Deactivate Workstation

This subroutine may be used to deactivate a workstation. When a workstation is inactive, output primitives and segments will not be written to it.

The calling sequence is:

```
CALL GDAWK(WKID)
```

The input parameter is:

- `WKID` An integer giving the workstation identifier.

2.1.5. Subroutine GCLRWK: Clear Workstation

This subroutine may be used to clear the display surface of a workstation. Graphic primitives outside of segments on the workstation are deleted, and all segments associated with the workstation are removed from the workstation. If a segment is no longer associated with any workstation, then the segment itself is deleted.

The calling sequence is:

```
CALL GCLRWK(WKID,COFL)
```

The input parameters are:

- WKID — An integer giving the workstation identifier.
- COFL — An integer giving a control flag. This value should be 0 (GCONDI) if the display is to be cleared only if it is not empty, and 1 (GALWAY) if it is always to be cleared. GKS-EZ suggests that a value of 1 (GALWAY) be used.

2.1.6. GKS Implementation

Subroutine GZOPWK is straightforward but lengthy. Among the operations it performs are:

1. If GKS is not open, it calls GOPKS. Unit 6 is used for the error message file, and the implementation default buffer size is used. If either of these is inappropriate, the user can call GOPKS before GZOPWK is called for the first time. However, if the user calls GOPKS, it is important that the other functions that GZOPWK performs at this time are done by the user.
2. If GKS is not open, the output attributes are set to their default values under GKS-EZ.
3. If GKS is not open, all of the normalization transformations are set to default values. Normalization transformation 1 is selected and its priority is raised. Clipping for the normalization transformation is also turned on.
4. The workstation is opened and activated.
5. If the workstation has output capability, the deferral mode is set to "before next interaction locally" (GBNIL) and the implicit regeneration mode is set to "allowed" (GALLOW).
6. If the workstation has output capability, the color table for the workstation is initialized. The appropriate color indices in the range 0 through 7 are initialized.
7. If the workstation has input capability, the mode of the input functions is set to request with echo turned on.

Subroutine GZCLWK is much simpler; it deactivates and closes the workstation. If no more workstations are open, it closes GKS itself.

Subroutines GACWK, GDAWK, and GCLRWK are native GKS subroutines.

The user should be aware that the number of workstations that can be open at one time is implementation-dependent. Therefore, a GKS program that opens

as few as two workstations at one time may not be transportable to another GKS implementation.

2.2. Output Functions

GKS-EZ provides five graphic primitives. These primitives are lines, markers, text, fill area, and extended text. The first four primitives are the usual GKS primitives. The fifth, extended text, is an additional primitive supplied by GKS-EZ. The attributes of the primitives can be set with the subroutines described in the section on output attributes. In particular, the color and geometric aspects of these primitives can be controlled.

2.2.1. Subroutine GPL: Polyline

This subroutine may be used to draw a sequence of concatenated straight line segments. The polyline begins at the first given point and proceeds through each of the succeeding points. The lines may be solid, dashed, dotted, or dot-dashed. Their width and color may also be controlled.

The calling sequence is:

```
CALL GPL(N,PXA,PYA)
```

The input parameters are:

N	An integer giving the number of points in the polyline.
PXA	A real array of dimension N that gives the x coordinates of the points in the polyline in world coordinates.
PYA	A real array of dimension N that gives the y coordinates of the points in the polyline in world coordinates.

2.2.2. Subroutine GPM: Polymarker

This subroutine may be used to draw a sequence of markers. A marker may consist of a single point or a more elaborate plotting symbol. If a plotting symbol is used, its size may be controlled.

The calling sequence is:

```
CALL GPM(N,PXA,PYA)
```

The input parameters are:

N	An integer giving the number of points in the polymarker.
PXA	A real array of dimension N that gives the x coordinates of the markers in world coordinates.
PYA	A real array of dimension N that gives the y coordinates of the markers in world coordinates.

2.2.3. Subroutine GTX: Text

This subroutine may be used to draw a string of characters. The size, orientation, and color of the characters may be controlled. The location point of the character string is quite flexible and may be any of the corners of the box enclosing the string as well as a number of other points. The characters that the user tries to draw with this subroutine should be limited to the ASCII character set.

The calling sequence is:

```
CALL GTX(PX,PY,CHARS)
```

The input parameters are:

- `PX` A real value that gives the x coordinate of the location point of the character string in world coordinates.
- `PY` A real array that gives the y coordinate of the location point of the character string in world coordinates.
- `CHARS` A character string containing the characters.

2.2.4. Subroutine GFA: Fill Area

This subroutine may be used to fill a polygonal area. The polygon is defined by giving its vertex points. The interior of the polygon may be hollow, solid, or filled with certain patterns. If the first and last of the given points are not the same, GKS will supply a closing point.

The calling sequence is:

```
CALL GFA(N,PXA,PYA)
```

The input parameters are:

- `N` An integer giving the number of points defining the fill area.
- `PXA` A real array of dimension `N` that gives the x coordinates of the points defining the fill area in world coordinates.
- `PYA` A real array of dimension `N` that gives the y coordinates of the points defining the fill area in world coordinates.

2.2.5. Subroutine GZET: Extended Text

This subroutine may be used to draw a string of characters. The characters produced by this subroutine are quite varied and include the upper and lower case Roman, Greek, Cyrillic, and Hebrew alphabets, and a wide variety of special characters. A versatile subscripting and superscripting ability is also available. A complete description of the characters that can be produced by this subroutine will be given in Chapter 3.

The characters drawn by this subroutine are specified by giving a pair of character strings of equal length. The actual character produced is determined by examining corresponding positions in the two strings. The first string, the primary

characters, gives an approximation to the actual character while the second string, the secondary characters, gives a modifier character. As an example, suppose the primary string is "AAA" and the secondary string is " LG." In this case, the first character drawn is an upper case Roman "A" (because the first secondary character is a blank), the second character is a lower case Roman "A" (because the second secondary character is an "L"), and the third character is a lower case Greek alpha (because the third secondary character is a "G").

All of these characters may be produced in any of three fonts. Two of these fonts, the *simplex* and *duplex* fonts, are drawn with polylines while the third, the *solid* font, is drawn with fill areas. The simplex font minimizes the complexity of the characters, while the duplex font has some of the properties of typeset characters. The solid font can be useful when large lettering is required.

The calling sequence is:

```
CALL GZET(PX,PY,PCHARS,SCHARS)
```

The input parameters are:

PX	A real value that gives the x coordinate of the location point of the character string in world coordinates.
PY	A real array that gives the y coordinate of the location point of the character string in world coordinates.
PCHARS	A character string containing the primary characters.
SCHARS	A character string containing the secondary characters.

2.2.6. GKS Implementation

Subroutines GPL, GPM, GTX, and GFA are all native GKS subroutines.

GZET first saves the current polyline or fill area attributes and replaces them with the attributes necessary to draw the characters. Next, it draws the actual characters using subroutine GPL or GFA. Finally, it restores the original attributes.

2.3. Output Attributes

Six subroutines may be used to set the attributes of the graphic primitives. The first five correspond to the five graphic primitives in the previous section. The sixth is used to set the pick identification of the graphic primitives; it is only needed when a program is using an interactive device with a pick control unit. When an attribute is set by any of these subroutines, it remains in effect until it is changed by a subsequent subroutine call. The attributes apply to all open workstations.

The first five subroutines all have two arguments, a character string and a real array. Attributes to be set are specified in the character string. If a value is required with the attribute, it may be given in the character string or in the real array. For example, the statement

```
CALL GZSPLA('RED,WIDTH=2.5',DUMMY)
```

sets the polyline color attribute to red and sets the line width to 2.5 times its normal value. All other attributes are unchanged. Notice that individual attributes in the character string are separated by commas. Exactly the same result is produced by

```
REAL       RA(8)
RA(3)=2.5
CALL GZSPLA('RED,WIDTH=@3',RA)
```

where the "@" character is used to indicate that the value is to be found in the specified element of the real array. The order of the items in the character string is usually not significant. However, the user should be very careful to spell the items in the character string correctly and to use upper case characters; invalid items cause an error message to be printed and all of the items in the string to be ignored.

2.3.1. Subroutine GZSPLA: Set Polyline Attributes

This subroutine may be used to set the polyline attributes of color, line structure, and line width.

The calling sequence is:

```
CALL GZSPLA(OPTN,RA)
```

The input parameters are:

OPTN — A character string which may contain any of the following items:

- DEFAULT: This item causes the polyline attributes to be set to their default values. This item will be executed first even if it is not the first item in the options list.
- BKGRND, NORMAL, RED, GREEN, BLUE, YELLOW, MAGENTA, or CYAN: These items control the color of the lines.
- SOLID, DASHED, DOTTED, or DASHDOT: These items control the type of the lines.
- WIDTH=⟨*width-value*⟩: This item controls the line width scale factor.

RA — A real array that may contain numeric values for some of the attributes.

The default values are NORMAL, SOLID, and WIDTH=1.0.

2.3.2. Subroutine GZSPMA: Set Polymarker Attributes

This subroutine may be used to set the polymarker attributes of color, marker type, and marker size.

The calling sequence is:

```
CALL GZSPMA(OPTN,RA)
```

The input parameters are:

OPTN — A character string which may contain any of the following items:

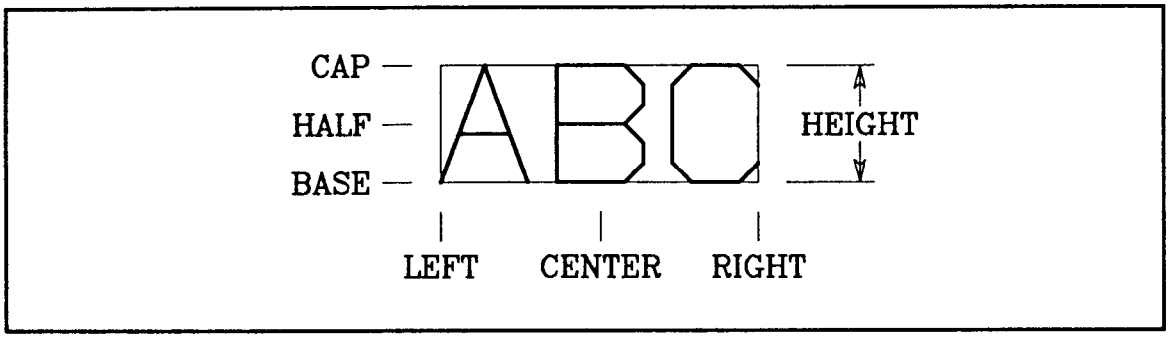

Figure 2.1. Text alignment and height

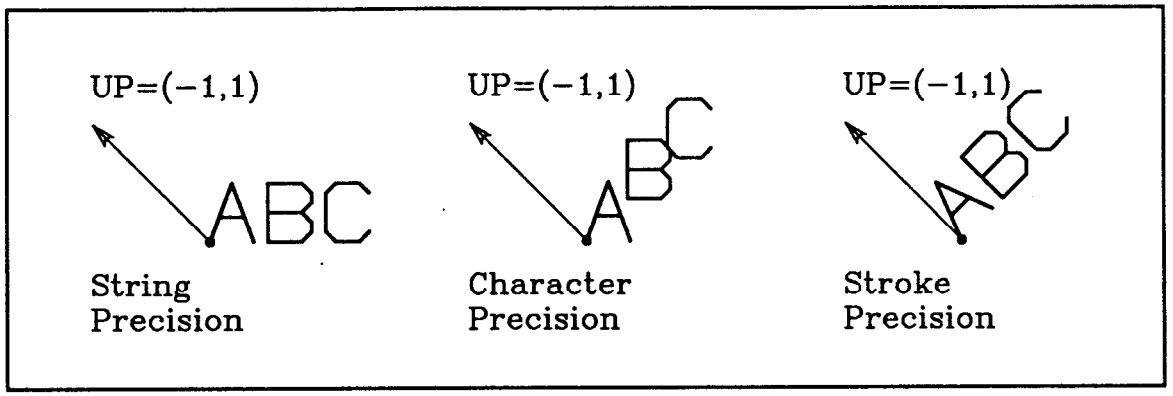

Figure 2.2. Character precision and the up vector

`DEFAULT`: This item causes the polymarker attributes to be set to their default values. This item will be executed first even if it is not the first item in the options list.

`BKGRND`, `NORMAL`, `RED`, `GREEN`, `BLUE`, `YELLOW`, `MAGENTA`, or `CYAN`: These items control the color of the markers.

`POINT`, `PLUS`, `AST`, `OMARK`, or `XMARK`: These items control the type of the markers.

`SIZE=`⟨*size-value*⟩: This item controls the marker size scale factor.

`RA` A real array that may contain numeric values for some of the attributes.

The default values are `NORMAL`, `POINT`, and `SIZE=1.0`.

2.3.3. Subroutine `GZSTXA`: Set Text Attributes

This subroutine may be used to set the text attributes, including color and a number of essentially geometric attributes. The height may be given, and the meaning of the x and y position supplied by subroutine `GTX` may be specified as shown in Figure (2.1). The spacing between adjacent characters is approximately the same as the character height. In addition, the character precision and orientation may be given as illustrated in Figure (2.2). String precision allows GKS to position the text in the most economical manner that the workstation allows. This usually means a

horizontal string of characters. In character precision, each character is individually positioned. In either string or character precision, the hardware character generator of the workstation will usually be used, and the height may not be matched very closely. In stroke precision, the characters will usually be drawn with individual line segments and both height and orientation will be matched as closely as possible. The up vector controls the orientation of the characters and its effect can vary with the precision as shown in Figure (2.2).

In string precision, GKS is not required to use many of the text attributes, such as text alignment, in drawing the characters and this action may be different on different workstations or different GKS implementations. For this reason, the use of string precision text is strongly discouraged. The user must also remember that a GKS implementation may upgrade the user specified precision from string to character, or from character to stroke, if it decides to do so.

The calling sequence is:

```
CALL GZSTXA(OPTN,RA)
```

The input parameters are:

`OPTN` A character string which may contain any of the following items:

- `DEFAULT`: This item causes the text attributes to be set to their default values. This item will be executed first even if it is not the first item in the options list.
- `BKGRND`, `NORMAL`, `RED`, `GREEN`, `BLUE`, `YELLOW`, `MAGENTA`, or `CYAN`: These items control the color of the text.
- `HEIGHT=`⟨*height-value*⟩: This item controls the height of the text.
- `LEFT`, `CENTER`, or `RIGHT`: These items control the horizontal alignment of the text.
- `BASE`, `HALF`, or `CAP`: These items control the vertical alignment of the text.
- `STRING`, `CHAR`, or `STROKE`: These items control the precision of the text.
- `UP=(`⟨*x-direction-value*⟩, ⟨*y-direction-value*⟩`)`: This item controls the up vector of the text.

`RA` A real array that may contain numeric values for some of the attributes.

The default values are `NORMAL`, `HEIGHT=0.01`, `LEFT`, `BASE`, `CHAR`, and `UP=(0.0,1.0)`.

2.3.4. Subroutine `GZSFAA`: Set Fill Area Attributes

This subroutine may be used to set the fill area attributes of color and interior style.

The calling sequence is:

```
CALL GZSFAA(OPTN,RA)
```

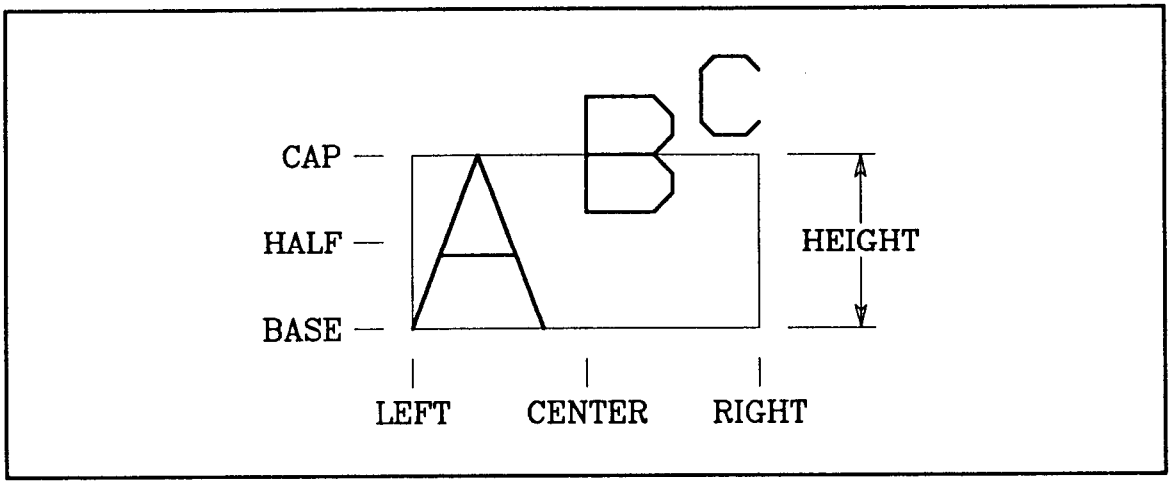

Figure 2.3. Height and alignment in the extended character sets

The input parameters are:

OPTN A character string which may contain any of the following items:

DEFAULT: This item causes the fill area attributes to be set to their default values. This item will be executed first even if it is not the first item in the options list.

BKGRND, NORMAL, RED, GREEN, BLUE, YELLOW, MAGENTA, or CYAN: These items control the color of the fill area.

HOLLOW, SOLID, PATTERN, or HATCH: These items control the fill area interior style.

RA A real array that may contain numeric values for some of the attributes.

The default values are NORMAL and SOLID.

2.3.5. Subroutine GZSETA: Set Extended Text Attributes

This subroutine may be used to set the extended text attributes. All of the attributes available for simple text, except the precision attribute, are available for extended text; the text drawn by this subroutine is always in stroke precision. In addition to the simple text attributes, the character font, mono-spacing versus proportional spacing, and the width of the polylines used to draw the characters in the simplex and duplex fonts may be specified.

Since superscripting and subscripting are allowed in the characters produced by this subroutine, the meaning of the height, and horizontal and vertical alignment specifications can be ambiguous. The problem and its solution are illustrated in Figure (2.3). The height is that of the first character and the width of the alignment box is determined by the farthest extent of the characters being produced. The mono-spacing of the characters is also violated when superscripting, subscripting, or size changes are used in the strings.

The calling sequence is:

```
CALL GZSETA(OPTN,RA)
```

The input parameters are:

OPTN A character string which may contain any of the following items:

- DEFAULT: This item causes the extended text attributes to be set to their default values. This item will be executed first even if it is not the first item in the options list.
- BKGRND, NORMAL, RED, GREEN, BLUE, YELLOW, MAGENTA, or CYAN: These items control the color of the extended text.
- HEIGHT=⟨*height-value*⟩: This item controls the height of the text.
- LEFT, CENTER, or RIGHT: These items control the horizontal alignment of the text.
- BASE, HALF, or CAP: These items control the vertical alignment of the text.
- UP=(⟨*x-direction-value*⟩, ⟨*y-direction-value*⟩): This item controls the up vector of the text.
- SIMPLEX, DUPLEX, or SOLID: These items control the font used to produce the text.
- MONO or PROP: These items control whether the text is mono-spaced or proportionally spaced.
- WIDTH=⟨*width-value*⟩: This item controls the line width scale factor of the lines used to draw the characters in the simplex and duplex fonts.

RA A real array that may contain numeric values for some of the attributes.

The default values are NORMAL, HEIGHT=0.01, LEFT, BASE, UP=(0.0,1.0), SIMPLEX, PROP, and WIDTH=1.0.

2.3.6. Subroutine GSPKID: Set Pick Identification

This subroutine may be used to set the pick identification. It can only be called while a segment is being constructed. This identification applies to all subsequently defined graphic primitives until it is changed.

The calling sequence is:

```
CALL GSPKID(PKID)
```

The input parameter is:

PKID An integer giving the pick identification.

The default value of the pick identification is 0.

2.3.7. GKS Implementation

The first five subroutines work by scanning the character string and determining the attributes to be changed. In the case of the first four subroutines, the proper

GKS subroutines are called to set the attribute. In the case of GZSETA, the attributes are saved in a COMMON block where subroutine GZET can obtain them. Because GZSETA saves its information in a COMMON block, it is fundamentally different from the other four subroutines. Native GKS subroutines may be used to retrieve the current settings of the attributes assigned by the other subroutines; GKS-EZ does not provide any means of retrieving the attributes set by GZSETA.

There are additional GKS attributes that GKS-EZ does not support. For example, there are additional alignment attributes available for text material, but these seem to be totally redundant for most applications. There are also text spacing and text expansion factors, but these seem unnecessary for the vast majority of applications. The text path attribute is potentially useful, but it loses its meaning when the superscripting and subscripting ability of subroutine GZET is considered. It was principally because of this problem that text path was dropped. Most GKS implementations will provide a number of different patterns and hatch styles. Unfortunately, none of these are standardized, and any use of them could make an application program difficult to transport to another GKS implementation. GKS-EZ recommends that only the default pattern and hatch styles be used.

GKS-EZ uses individual attributes only; it never uses bundled attributes. Individual attributes in GKS are much simpler to use and are more forgiving. The use of bundled attributes results in an extra level of indirectness in the assignment of attributes, resulting in extremely obscure application programs. In addition, the number of attribute bundles available can vary from one GKS implementation to another, so using them can make a program implementation-dependent.

Subroutine packages similar to GKS-EZ have been criticized for the small number of colors that they support. In the first place, GKS-EZ is designed to run on a wide variety of graphic devices and inexpensive terminals do not usually have that much flexibility in this area. Secondly, even if they have great flexibility in this area, that flexibility is wasted when line drawn pictures are produced. A number of studies have been made which show that only a few intensity levels or colors can be reliably distinguished in line drawn pictures. For example, a table is shown in *Human Factors Problems in Computer-Generated Graphic Displays* [Bar66] which indicates that, with 95 percent accuracy, most people can distinguish only 4 different intensity levels and 11 different colors. In addition, to get results as high as 4 and 11, the intensity levels and colors must be chosen very carefully. A similar table in the article "The art of natural graphic man-machine conversation" [Fol74] gives even smaller numbers for easily distinguished attributes (2 intensity levels and 6 colors). However, it must be pointed out that the above conclusions only apply to graphic devices used to display line drawn pictures. When photographic-like images are produced, a large number of intensity levels and colors are required. This type of image can be produced in GKS using the cell array primitive. However, these programs are very device-dependent; the limits of the colors that can be produced, as well as the pixel size of the screen, must be taken into consideration. Programs that produce these kinds of pictures will have to be written in GKS itself. However, the real reason GKS-EZ only supports these few colors is that the structure of GKS makes any more general specification both device and implementation dependent.

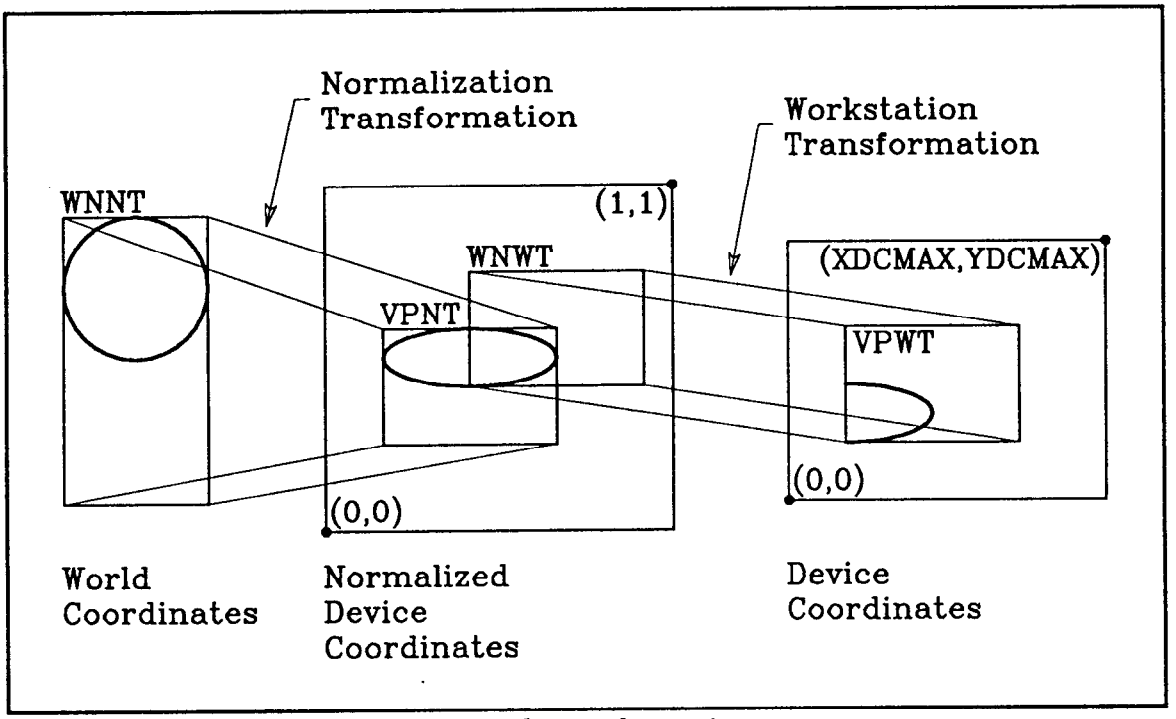

Figure 2.4. Coordinate systems and transformations

The defaults for these attributes are set when subroutine GZOPWK is called, and it in turn has to call subroutine GOPKS to initialize GKS.

Subroutine GSPKID is a native GKS subroutine.

2.4. Transformation Functions

The transformation from the world coordinate system of the user to the device coordinates of the workstation is handled in two steps. The first step, the normalization transformation, is a mapping from the world coordinate system to normalized device coordinates. The second step, the workstation transformation, transforms from normalized device coordinates to device coordinates. Each of these transformations is specified by giving a rectangular window in the source coordinate system and a rectangular viewport in the target coordinate system. The window and viewport of the workstation transformation must have the same aspect ratio. These concepts are illustrated in Figure (2.4). The useful limits of normalized device coordinates are from 0.0 to 1.0 in both x and y. The upper limits of the device coordinates may be obtained with subroutine GZQDSP. That subroutine is described in the section on inquiry functions.

GKS-EZ allows a separate workstation transformation for each workstation and a number of normalization transformations that is determined by the GKS implementation is use. GKS-EZ supplies a subroutine, GZQMNT, which returns the number of normalization transformations that are available. GZQMNT is described in the section on inquiry functions. The best practice is to define all of the normalization

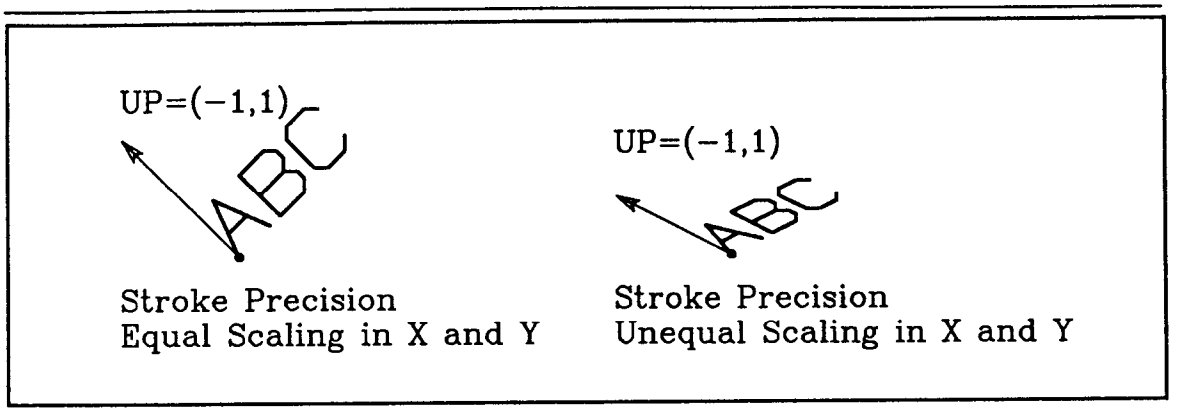

Figure 2.5. How scaling affects text

transformations and the workstation transformation in the beginning before any graphic primitives are sent to the workstation. On some devices, for some implementations, changing a transformation will change the graphic primitives that were defined under that transformation.

The windows and viewports are given by real arrays of dimension (2,2). For example, the array defining the window for the normalization transformation is given by

$$\texttt{WNNT} = \begin{pmatrix} \texttt{WNNT}(1,1) & \texttt{WNNT}(1,2) \\ \texttt{WNNT}(2,1) & \texttt{WNNT}(2,2) \end{pmatrix} = \begin{pmatrix} x_{min} & y_{min} \\ x_{max} & y_{max} \end{pmatrix}.$$

The default normalization transformations are defined by a window in the world coordinate system consisting of a unit square, that is, a square with both x and y ranging from 0.0 to 1.0, and a viewport in normalized device coordinates also consisting of a unit square. In GKS-EZ, the default normalization transformation is transformation number 1. The default workstation transformation is defined by a window in normalized device coordinates consisting of a unit square and a viewport consisting of a maximal square in the lower left of the device coordinate system; that is, a square with x and y ranging from 0.0 to `MIN(XDCMAX,YDCMAX)`.

There is, however, a major problem with the drawing of stroke precision text when the user modifies these transformations. The problem applies to stroke precision text produced by subroutine `GTX` and to all text produced by subroutine `GZET`. If the scaling, that is, the number of units in the world coordinate system per meter in device coordinates, is not the same in x and y, then the text will be distorted as illustrated in Figure (2.5). This distorted text is almost never what the user really wants. Nevertheless, it is the user's responsibility to assure that the scaling in x and y is the same in both directions when text material is produced. Since GKS constrains the workstation transformation to have equal aspect ratios in its window and viewport, equal scaling in x and y is assured if the window and viewport of the normalization transformation have equal aspect ratios. Because GKS may upgrade the precision of any text to stroke precision, this problem must always be considered. If the default transformations are used, the text will be undistorted.

2.4.1. Subroutine GZSNT: Set Normalization Transformation

This subroutine may be used to define the window and viewport of a normalization transformation. The normalization transformation maps from world coordinates to normalized device coordinates.

The calling sequence is:

```
CALL GZSNT(TNR,WNNT,VPNT)
```

The input parameters are:

- **TNR** An integer giving the number of the normalization transformation to be changed.
- **WNNT** A real array of dimension (2,2) that contains the window of the normalization transformation in world coordinates.
- **VPNT** A real array of dimension (2,2) that contains the viewport of the normalization transformation in normalized device coordinates.

2.4.2. Subroutine GZSWT: Set Workstation Transformation

This subroutine may be used to define the window and viewport of the workstation transformation. The workstation transformation maps from normalized device coordinates to device coordinates. The window and viewport of this transformation must have the same aspect ratio.

The calling sequence is:

```
CALL GZSWT(WKID,WNWT,VPWT)
```

The input parameters are:

- **WKID** An integer giving the workstation identifier.
- **WNWT** A real array of dimension (2,2) that contains the window of the workstation transformation in normalized device coordinates.
- **VPWT** A real array of dimension (2,2) that contains the viewport of the workstation transformation in device coordinates.

2.4.3. Subroutine GSELNT: Select Normalization Transformation

This subroutine may be used to select a normalization transformation. Subsequently generated graphic primitives are transformed by this transformation until another transformation is selected.

The calling sequence is:

```
CALL GSELNT(TNR)
```

The input parameter is:

- **TNR** An integer giving the number of the normalization transformation being selected.

2.4.4. GKS Implementation

The first two GKS-EZ subroutines are quite simple; they just call the pair of GKS subroutines that supply the window and viewport of the transformation separately. In the case of the normalization transformation, GKS-EZ only uses normalization transformation 1 through **MAXTNR** where **MAXTNR** is returned by subroutine **GZQMNT**. There is a normalization transformation with a number of 0 but that transformation cannot be changed.

Subroutine **GSELNT** is a native GKS subroutine.

2.5. Segment Functions

Three subroutines may be used to create and manipulate segments in GKS-EZ. These operations are usually only necessary when interactive workstations are being used. Using these subroutines, an application program may collect groups of output primitives together into segments. These segments are written to all active workstations. They may then be manipulated in a number of ways; the visibility, highlighting, or pick detectability may be changed, and the segment itself may be deleted.

It is invalid to try to create a new segment before a previous segment has been closed.

2.5.1. Subroutine GCRSG: Create Segment

This subroutine may be used to begin a segment. All subsequent output primitives until the segment is closed will belong to that segment.

The calling sequence is:

```
CALL GCRSG(SGNA)
```

The input parameter is:

SGNA An integer giving the identification of the segment being created. This identification must be different from any other existing segment.

2.5.2. Subroutine GCLSG: Close Segment

This subroutine may be used to close the current segment. Any subsequent output primitives will not belong to any segment.

The calling sequence is:

```
CALL GCLSG
```

This subroutine does not have any parameters.

2.5.3. Subroutine GZMNSG: Manipulate Segment

This subroutine may be used to manipulate a segment. Multiple operations

may be performed on a segment with a single call to this subroutine. For example, visibility, highlighting, and detectability can all be changed with a single statement.

The calling sequence is:

```
CALL GZMNSG(OPTN,SGNA)
```

The input parameters are:

OPTN A character string which may contain any of the following items:

- VISI or INVIS: These items can modify the visibility of the segment.
- NORMAL or HILITE: These items can modify the highlighting of the segment. Highlighting is an implementation-dependent function.
- UNDET or DETEC: These items can modify the pick detectability of the segment.
- DELETE: This item indicates that the segment is to be deleted.

SGNA An integer giving the identification of the segment being modified.

The default attributes, when a segment is initially created, are VISI, NORMAL, and UNDET.

2.5.4. GKS Implementation

Subroutines GCRSG and GCLSG are native GKS subroutines.

Subroutine GZMNSG works by scanning the character string and determining the properties to be changed. The proper GKS subroutines are then called to set the attributes or delete the segment.

2.6. Input Functions

The subroutines in this section are only needed when interactive application programs are being prepared. These subroutines use the interactive control units on the workstation to synchronize the operator actions with the program. When one of these subroutines is called, the program will halt and go into a wait state until the operator responds. The operator may either supply the requested information or abort the request.

GKS usually guarantees that an interactive workstation has at least one control unit of each type. If the workstation does not have a physical input device of the required type, GKS will usually simulate it, and that simulation will be workstation-dependent. GKS-EZ strongly recommends that only control unit 1 be used. If the control unit is real, the number available is device-dependent; if it is simulated, the number is implementation-dependent. Therefore, any use of control unit numbers larger than 1 can result in non-transportable application programs.

Each of the subroutines described in this section returns an integer status value.

2.6.1. Subroutine GRQLC: Request Locator

This subroutine may be used to request interaction with the locator control unit. The subroutine will return the x and y coordinates of a point in the world coordinate system, using the current normalization and workstation transformations.

The calling sequence is:

```
CALL GRQLC(WKID,LCDNR,STAT,TNR,PX,PY)
```

The input parameters are:

- `WKID` An integer giving the workstation identifier.
- `LCDNR` An integer giving the locator device number.

The output parameters are:

- `STAT` An integer giving the status of the request. This will be 0 (`GNONE`) if the operator declined to supply a locator position and 1 (`GOK`) if a locator position is available.
- `TNR` An integer giving the normalization transformation number.
- `PX` A real value that gives the x coordinate of the locator position in world coordinates.
- `PY` A real value that gives the y coordinate of the locator position in world coordinates.

2.6.2. Subroutine GRQSK: Request Stroke

This subroutine may be used to request interaction with the stroke control unit. The subroutine will return the x and y coordinates of a sequence of points in the world coordinate system, using the current normalization and workstation transformations.

The calling sequence is:

```
CALL GRQSK(WKID,SKDNR,N,STAT,TNR,NP,PXA,PYA)
```

The input parameters are:

- `WKID` An integer giving the workstation identifier.
- `SKDNR` An integer giving the stroke device number.
- `N` An integer giving the maximum number of points that can be accepted.

The output parameters are:

- `STAT` An integer giving the status of the request. This will be 0 (`GNONE`) if the operator declined to supply a stroke and 1 (`GOK`) if a stroke is available.
- `TNR` An integer giving the normalization transformation number.
- `NP` An integer giving the number of points actually returned.
- `PXA` A real array that gives the x coordinates of the points in the stroke in world coordinates.
- `PYA` A real array that gives the y coordinates of the points in the stroke in world coordinates.

2.6.3. Subroutine GRQVL: Request Valuator

This subroutine may be used to request interaction with the valuator control unit. The subroutine will return a number corresponding to the setting of the valuator. The range of values that may be returned is workstation-dependent.

The calling sequence is:

```
CALL GRQVL(WKID,VLDNR,STAT,VAL)
```

The input parameters are:

- WKID An integer giving the workstation identifier.
- VLDNR An integer giving the valuator device number.

The output parameters are:

- STAT An integer giving the status of the request. This will be 0 (GNONE) if the operator declined to supply a valuator value and 1 (GOK) if a valuator value is available.
- VAL A real value that gives the value of the valuator.

2.6.4. Subroutine GRQCH: Request Choice

This subroutine may be used to request interaction with the choice control unit. The subroutine will return a non-negative integer to represent the choice that was made. The range of values that can be returned is workstation-dependent and implementation-dependent.

The calling sequence is:

```
CALL GRQCH(WKID,CHDNR,STAT,CHNR)
```

The input parameters are:

- WKID An integer giving the workstation identifier.
- CHDNR An integer giving the choice device number.

The output parameters are:

- STAT An integer giving the status of the request. This will be 0 (GNONE) if the operator declined to supply a choice number, 1 (GOK) if a choice number is available, and 2 (GNCHOI) if no choice control unit is available.
- CHNR An integer giving the choice number.

2.6.5. Subroutine GRQPK: Request Pick

This subroutine may be used to request interaction with the pick control unit. The subroutine will return the identification of the segment and the identification of the graphic primitive within the segment.

The calling sequence is:

```
CALL GRQPK(WKID,PKDNR,STAT,SGNA,PKID)
```

The input parameters are:

`WKID` An integer giving the workstation identifier.

`PKDNR` An integer giving the pick device number.

The output parameters are:

`STAT` An integer giving the status of the request. This will be 0 (`GNONE`) if the operator declined to supply a pick value, 1 (`GOK`) if a pick value is available, and 2 (`GNPICK`) if no pick control unit is available.

`SGNA` An integer giving the segment name of the selected item.

`PKID` An integer giving the pick identification of the selected item.

2.6.6. Subroutine `GRQST`: Request String

This subroutine may be used to request interaction with the string control unit. The subroutine will return a string of characters.

The calling sequence is:

```
CALL GRQST(WKID,STDNR,STAT,LOSTR,STR)
```

The input parameters are:

`WKID` An integer giving the workstation identifier.

`STDNR` An integer giving the string device number.

The output parameters are:

`STAT` An integer giving the status of the request. This will be 0 (`GNONE`) if the operator declined to supply a string and 1 (`GOK`) if a string is available.

`LOSTR` An integer giving the number of characters returned.

`STR` A character string that contains the string being supplied.

2.6.7. GKS Implementation

All of these subroutines are native GKS subroutines.

The GKS-EZ subroutine `GZOPWK` initializes all INPUT or OUTIN workstations to request mode with the default echo turned on. GKS-EZ suggests the use of the request mode for interaction because it is the simplest of the three available modes. In addition, the sample and event forms of input do not become available until the input level of GKS is raised to its maximum value. GKS-EZ also suggests that the user accept the default prompt, echo type, and echo area of the underlying GKS in use. Changing these values can introduce device or implementation dependencies.

2.7. Inquiry Functions

GKS-EZ provides two inquiry functions. They are useful when normalization or workstations transformations must be manipulated.

2.7.1. Subroutine GZQMNT: Inquire Number of Normalization Transformations

This subroutine may be used to obtain the number of normalization transformations that are available. This information is necessary when the normalization transformation is to be manipulated.

The calling sequence is:

```
CALL GZQMNT(MAXTNR)
```

The output parameter is:

MAXTNR An integer giving the maximum normalization transformation number.

2.7.2. Subroutine GZQDSP: Inquire Display Space Size

This subroutine may be used to obtain the extent of device coordinates for a specific type of workstation. This information is necessary if the workstation transformation is to be manipulated.

The calling sequence is:

```
CALL GZQDSP(WTYPE,DCUNIT,XDCMAX,YDCMAX)
```

The input parameter is:

WTYPE An integer giving the workstation type. This is the same value that was supplied to subroutine GZOPWK.

The output parameters are:

DCUNIT An integer giving the coordinate units being reported. The value will be 0 (GMETRE) if the units are in meters and 1 (GOTHU) if some other measure is being used.

XDCMAX A real value that gives the maximum x coordinate in device coordinates.

YDCMAX A real value that gives the maximum y coordinate in device coordinates.

2.7.3. GKS Implementation

These subroutines are really very straightforward. They simply call GKS subroutines and discard some of the unnecessary information supplied by those subroutines.

GKS supplies a very large number of inquiry subroutines. However, most of these are redundant for simple applications because they either return program states that the user has previously set, or they return device-dependent information that most programs should avoid.

Chapter 3

The Extended Character Set

This section defines all of the characters that may be produced by subroutine GZET. The characters consist of the upper and lower case Roman, Greek, Cyrillic, and Hebrew alphabets, the numerals, and a great variety of special characters. In addition, a flexible position and size control scheme, including subscripting and superscripting, is provided. The extended character set may be produced in any of three fonts. Two of these fonts, the simplex and duplex fonts, are drawn with polylines while the third, the solid font, is drawn with fill areas. The simplex font minimizes the number of strokes in the character, while the duplex font produces characters that have the appearance of typeset characters. The solid font can be useful when large lettering is required. The full extended character set is described in the following table. The table gives the primary and secondary character followed by its description. The symbol "␣" stands for a blank.

The Upper Case Roman Alphabet:

- A␣ Upper case Roman A
- B␣ Upper case Roman B
- C␣ Upper case Roman C
- D␣ Upper case Roman D
- E␣ Upper case Roman E
- F␣ Upper case Roman F
- G␣ Upper case Roman G
- H␣ Upper case Roman H
- I␣ Upper case Roman I
- J␣ Upper case Roman J
- K␣ Upper case Roman K
- L␣ Upper case Roman L
- M␣ Upper case Roman M
- N␣ Upper case Roman N
- O␣ Upper case Roman O
- P␣ Upper case Roman P
- Q␣ Upper case Roman Q
- R␣ Upper case Roman R
- S␣ Upper case Roman S
- T␣ Upper case Roman T
- U␣ Upper case Roman U
- V␣ Upper case Roman V

W␣ Upper case Roman W
X␣ Upper case Roman X
Y␣ Upper case Roman Y
Z␣ Upper case Roman Z

The Lower Case Roman Alphabet:

AL Lower case Roman A
BL Lower case Roman B
CL Lower case Roman C
DL Lower case Roman D
EL Lower case Roman E
FL Lower case Roman F
GL Lower case Roman G
HL Lower case Roman H
IL Lower case Roman I
JL Lower case Roman J
KL Lower case Roman K
LL Lower case Roman L
ML Lower case Roman M
NL Lower case Roman N
OL Lower case Roman O
PL Lower case Roman P
QL Lower case Roman Q
RL Lower case Roman R
SL Lower case Roman S
TL Lower case Roman T
UL Lower case Roman U
VL Lower case Roman V
WL Lower case Roman W
XL Lower case Roman X
YL Lower case Roman Y
ZL Lower case Roman Z

Upper Case Auxiliary Roman Characters:

10 Upper case Latin and Scandinavian ligature AE
D0 Upper case Icelandic Eth
L0 Upper case Polish suppressed L
O0 Upper case Scandinavian O with slash
20 Upper case French ligature OE
T0 Upper case Icelandic Thorn

Lower Case Auxiliary Roman Characters:

A1 Lower case alternate Roman A
11 Lower case Latin and Scandinavian ligature AE

D1 Lower case Icelandic Eth
31 Lower case Roman ligature FF
41 Lower case Roman ligature FI
51 Lower case Roman ligature FL
61 Lower case Roman ligature FFI
71 Lower case Roman ligature FFL
G1 Lower case alternate Roman G
I1 Lower case dotless Roman I
J1 Lower case dotless Roman J
L1 Lower case Polish suppressed L
O1 Lower case Scandinavian O with slash
21 Lower case French ligature OE
S1 Lower case German double S
T1 Lower case Icelandic Thorn

The Upper Case Greek Alphabet:

AF Upper case Greek Alpha
BF Upper case Greek Beta
GF Upper case Greek Gamma
DF Upper case Greek Delta
EF Upper case Greek Epsilon
ZF Upper case Greek Zeta
HF Upper case Greek Eta
QF Upper case Greek Theta
IF Upper case Greek Iota
KF Upper case Greek Kappa
LF Upper case Greek Lambda
MF Upper case Greek Mu
NF Upper case Greek Nu
XF Upper case Greek Xi
OF Upper case Greek Omicron
PF Upper case Greek Pi
RF Upper case Greek Rho
SF Upper case Greek Sigma
TF Upper case Greek Tau
UF Upper case Greek Upsilon
FF Upper case Greek Phi
CF Upper case Greek Chi
YF Upper case Greek Psi
WF Upper case Greek Omega

The Lower Case Greek Alphabet:

AG Lower case Greek Alpha
BG Lower case Greek Beta

GG Lower case Greek Gamma
DG Lower case Greek Delta
EG Lower case Greek Epsilon
ZG Lower case Greek Zeta
HG Lower case Greek Eta
QG Lower case Greek Theta
IG Lower case Greek Iota
KG Lower case Greek Kappa
LG Lower case Greek Lambda
MG Lower case Greek Mu
NG Lower case Greek Nu
XG Lower case Greek Xi
OG Lower case Greek Omicron
PG Lower case Greek Pi
RG Lower case Greek Rho
SG Lower case Greek Sigma
TG Lower case Greek Tau
UG Lower case Greek Upsilon
FG Lower case Greek Phi
CG Lower case Greek Chi
YG Lower case Greek Psi
WG Lower case Greek Omega
1G Lower case Greek Epsilon (variant)
2G Lower case Greek Theta (variant)
3G Lower case Greek Pi (variant)
4G Lower case Greek Rho (variant)
5G Lower case Greek Sigma (variant)
6G Lower case Greek Phi (variant)

The Upper Case Cyrillic Alphabet:

AB Upper case Cyrillic Ah
BB Upper case Cyrillic Beh
VB Upper case Cyrillic Veh
GB Upper case Cyrillic Geh
DB Upper case Cyrillic Deh
EB Upper case Cyrillic Yeh
XB Upper case Cyrillic Zheh
ZB Upper case Cyrillic Zeh
IB Upper case Cyrillic Ee
1B Upper case Cyrillic Ee S Kratkoy
KB Upper case Cyrillic Kah
LB Upper case Cyrillic El
MB Upper case Cyrillic Em
NB Upper case Cyrillic En

OB Upper case Cyrillic Oh
PB Upper case Cyrillic Peh
RB Upper case Cyrillic Err
SB Upper case Cyrillic Ess
TB Upper case Cyrillic Teh
UB Upper case Cyrillic Ooh
FB Upper case Cyrillic Ef
HB Upper case Cyrillic Kha
CB Upper case Cyrillic Tseh
2B Upper case Cyrillic Cheh
3B Upper case Cyrillic Shah
4B Upper case Cyrillic Shchah
QB Upper case Cyrillic Tvyordy Znak
YB Upper case Cyrillic Yery
5B Upper case Cyrillic Myakhki Znak
6B Upper case Cyrillic Eh Oborotnoye
WB Upper case Cyrillic Yoo
JB Upper case Cyrillic Yah

The Lower Case Cyrillic Alphabet:

AC Lower case Cyrillic Ah
BC Lower case Cyrillic Beh
VC Lower case Cyrillic Veh
GC Lower case Cyrillic Geh
DC Lower case Cyrillic Deh
EC Lower case Cyrillic Yeh
XC Lower case Cyrillic Zheh
ZC Lower case Cyrillic Zeh
IC Lower case Cyrillic Ee
1C Lower case Cyrillic Ee S Kratkoy
KC Lower case Cyrillic Kah
LC Lower case Cyrillic El
MC Lower case Cyrillic Em
NC Lower case Cyrillic En
OC Lower case Cyrillic Oh
PC Lower case Cyrillic Peh
RC Lower case Cyrillic Err
SC Lower case Cyrillic Ess
TC Lower case Cyrillic Teh
UC Lower case Cyrillic Ooh
FC Lower case Cyrillic Ef
HC Lower case Cyrillic Kha
CC Lower case Cyrillic Tseh
2C Lower case Cyrillic Cheh

3C Lower case Cyrillic Shah
4C Lower case Cyrillic Shchah
QC Lower case Cyrillic Tvyordy Znak
YC Lower case Cyrillic Yery
5C Lower case Cyrillic Myakhki Znak
6C Lower case Cyrillic Eh Oborotnoye
WC Lower case Cyrillic Yoo
JC Lower case Cyrillic Yah

The Hebrew Alphabet:

AH Hebrew Aleph
BH Hebrew Beth
GH Hebrew Gimel
DH Hebrew Daleth
HH Hebrew He
VH Hebrew Vav
ZH Hebrew Zayin
CH Hebrew Cheth
OH Hebrew Teth
YH Hebrew Yod
KH Hebrew Kaph
LH Hebrew Lamed
MH Hebrew Mem
NH Hebrew Nun
SH Hebrew Sameth
XH Hebrew Ayin
PH Hebrew Pe
EH Hebrew Sadhe
QH Hebrew Koph
RH Hebrew Resh
WH Hebrew Sin/Shin
TH Hebrew Tav
1H Hebrew Kaph (end of word)
2H Hebrew Men (end of word)
3H Hebrew Nun (end of word)
4H Hebrew Pe (end of word)
5H Hebrew Sadhe (end of word)

The Numerals:

0␣ Numeral 0
1␣ Numeral 1
2␣ Numeral 2
3␣ Numeral 3
4␣ Numeral 4

5␣ Numeral 5
6␣ Numeral 6
7␣ Numeral 7
8␣ Numeral 8
9␣ Numeral 9

Common Special Symbols:

␣␣ Blank
+␣ Plus sign
-␣ Minus sign
*␣ Asterisk
/␣ Slash mark
=␣ Equal sign
.␣ Period
,␣ Comma
(␣ Left parenthesis
)␣ Right parenthesis

Special Symbols for Punctuation:

.P Colon
,P Semi-colon
EP Exclamation mark
UP Question mark
IP Interrobang
FP Inverted exclamation
VP Inverted question
AP Apostrophe
QP Quotation marks
OP Single left quote
1P Single right quote
2P Double left quote
3P Double right quote
SP New section
PP New paragraph or Pilcrow sign
DP Dagger
RP Double dagger

Additional Special Symbols:

DS Dollar sign
CS Cent sign
SS British Sterling
YS Japanese Yen
QS International currency symbol
+S Ampersand

PS	Pound sign
AS	At sign
OS	Copyright
GS	Registered
0S	Percent sign
1S	Per thousand sign
VS	Vertical line
IS	Broken vertical line
WS	Double vertical line
US	Underline
NS	Not sign
/S	Backwards slash
(S	Left bracket
)S	Right bracket
LS	Left brace
RS	Right brace
BS	Left angle bracket
ES	Right angle bracket
XS	Accent mark
TS	Caret mark

Mathematical Special Symbols:

.M	Dot product
XM	Cross product
/M	Division sign
PM	Group plus
*M	Group multiply
+M	Plus or minus
-M	Minus or plus
AM	And
VM	Or
UM	Therefore
WM	Since
LM	Less than
GM	Greater than
MM	Less than or equal
HM	Greater than or equal
3M	Much less than
4M	Much greater than
NM	Not equal
=M	Identically equal
KM	Approximately equal
CM	Congruent to
SM	Similar to

FM	Approximate
RM	Proportional to
TM	Perpendicular to
2M	Surd
DM	Degrees
IM	Integral sign
JM	Line integral
YM	Partial derivative
ZM	Del
(M	Left floor bracket
)M	Right floor bracket
BM	Left ceiling bracket
EM	Right ceiling bracket
0M	Infinity

Set Theoretic Special Symbols:

ET	Existential quantifier
AT	Universal quantifier
MT	Membership symbol
NT	Membership negation
IT	Intersection
UT	Union
LT	Contained in
GT	Contains
KT	Contained in or equals
FT	Contains or equals

Physics Special Symbols:

HK	H-bar
LK	Lambda-bar

Astronomical Special Symbols:

HA	Sun
MA	Mercury
VA	Venus
EA	Earth
WA	Mars
JA	Jupiter
SA	Saturn
UA	Uranus
NA	Neptune
PA	Pluto
0A	Moon
CA	Comet

- `*A` Star
- `XA` Ascending node
- `YA` Descending node
- `KA` Conjunction
- `QA` Quadrature
- `TA` Opposition
- `0A` Aries
- `1A` Taurus
- `2A` Gemini
- `3A` Cancer
- `4A` Leo
- `5A` Virgo
- `6A` Libra
- `7A` Scorpius
- `8A` Sagittarius
- `9A` Capricornus
- `AA` Aquarius
- `BA` Pisces

Drawing Symbols, Arrows, and Pointers:

- `0W` Underscore
- `1W` Midscore
- `2W` Overscore
- `UW` Up arrow
- `DW` Down arrow
- `LW` Left arrow
- `RW` Right arrow
- `BW` Left/right arrow

Diacritical Marks:

- `GD` Grave accent
- `AD` Acute accent
- `HD` Hat or circumflex
- `TD` Tilde or squiggle
- `MD` Macron or bar
- `BD` Breve accent
- `DD` Dot accent
- `UD` Umlaut or dieresis
- `RD` Ring or circle
- `VD` Caron, hacek, or check
- `LD` Long Hungarian umlaut
- `WD` Over arrow
- `CD` Cedilla accent
- `-D` Under bar

.D Under dot
,D Under dots
PD Prime

Horizontal and Vertical Movement Control:

␣ U Null
0U Backwards blank
1U Half blank
2U Half backwards blank
3U Third blank
4U Third backwards blank
5U Sixth blank
6U Sixth backwards blank
1V Half up movement
2V Half down movement
3V Third up movement
4V Third down movement
5V Sixth up movement
6V Sixth down movement

Subscript and Superscript Control:

0X Enter subscript mode
1X Leave subscript mode
2X Enter superscript mode
3X Leave superscript mode

Character Size Control:

0Y Increase size by one-half
1Y Decrease size by one-third
2Y Increase size by one-third
3Y Decrease size by one-fourth
4Y Increase size by one-sixth
5Y Decrease size by one-seventh

Position Control:

0Z Put current state in first save area
1Z Restore state from first save area
2Z Put current state in second save area
3Z Restore state from second save area
4Z Put current state in third save area
5Z Restore state from third save area
6Z Put current state in fourth save area
7Z Restore state from fourth save area

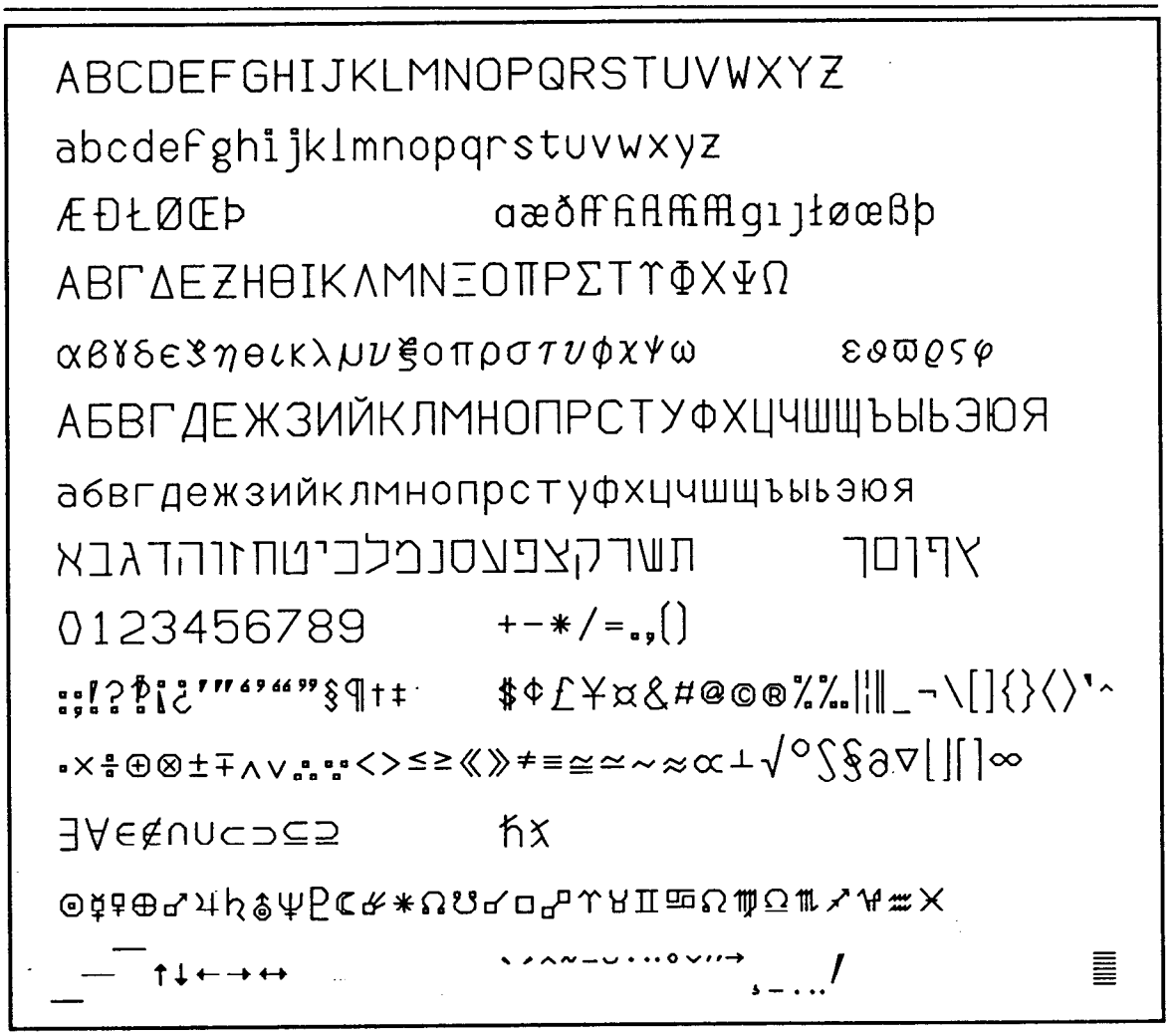

Figure 3.1. The simplex font of the extended character set

In addition to the primary and secondary character pairs shown above, most of the printable characters in the ASCII character set as described in *American National Standard for Information Systems: Coded Character Sets, 7-bit American National Standard Code for Information Interchange (7-bit ASCII)* [ANS86] will be produced with a secondary character of blank. Thus, if the primary character is a lower case Roman letter and the secondary character is a blank, then the proper character will be produced. The user, however, is encouraged to use the character pairs given in the above tables. The use of these character pairs will enhance the portability of the application program to non-ASCII computers.

The underscore, midscore, and overscore characters in the above table have some special properties. The purpose of these characters is to allow the programmer to draw lines under or over a line of text. Two consecutive underscore characters, for example, will join together into a single line (this is not true of the underline character). Thus the programmer, with some difficulty, can generate such things as fractions. The overscore will also join properly with the surd character to form a

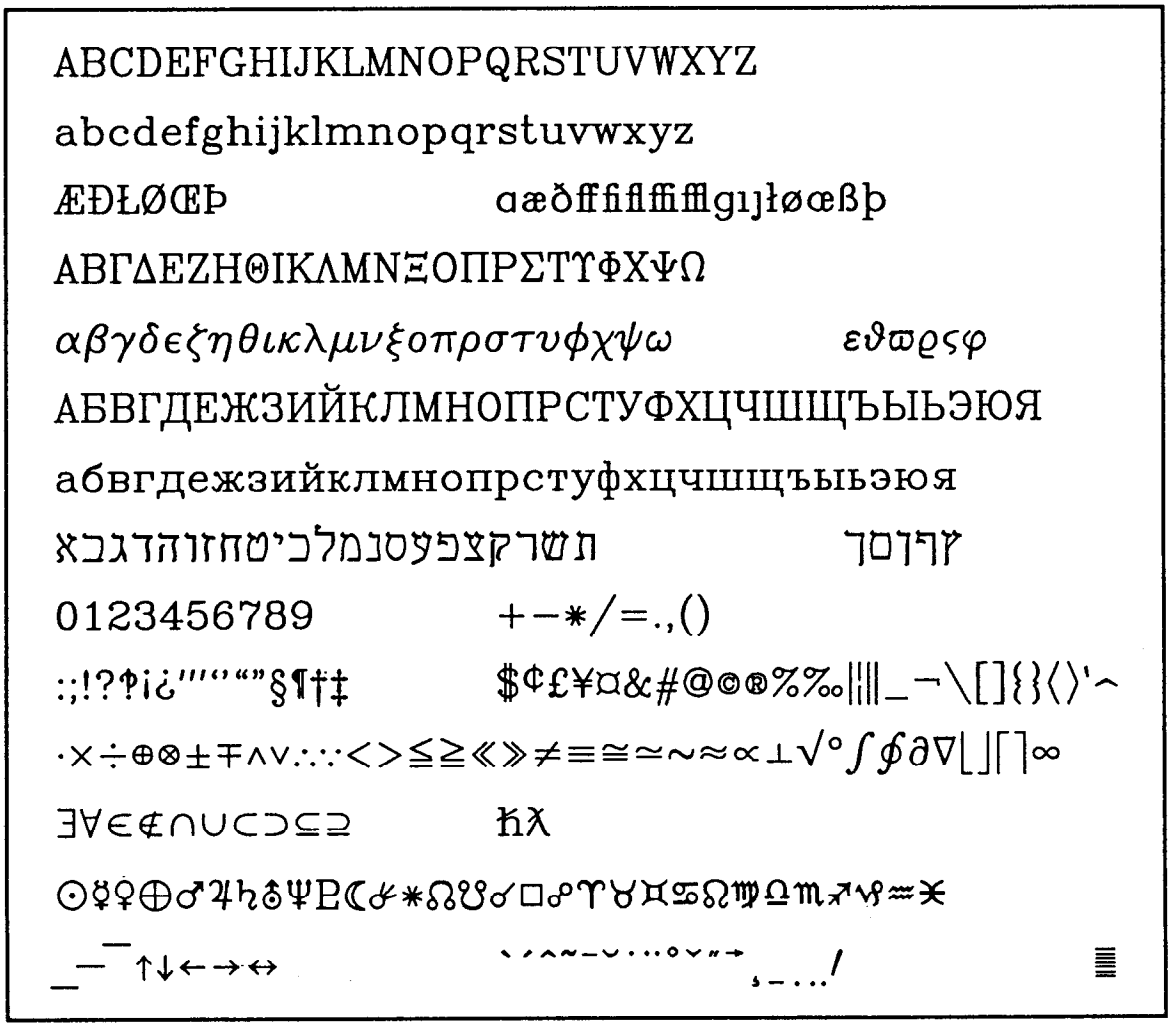

Figure 3.2. The duplex font of the extended character set

full radical sign.

The diacritical marks may be used immediately following any drawn character or a full sized blank. When this is done, the mark will attach itself to the preceding character and will be centered on that character. The prime mark is different than the others. The prime is normally used as a superscript on another symbol. More than one prime may be used in a superscript and the spacing will be appropriately close. However, this may mean that a partial space will have to be inserted if something follows a prime.

After a character is drawn, it is always followed by a short blank space before the next character is drawn. When the character is a full blank, it produces a space representing the blank and then the blank space that follows all characters. The fractional blanks refer only to the space that represents the space itself. The backwards blanks cause exactly enough movement to eliminate the space representing the blank and its following space. Thus, a "third blank" followed by a "third backwards blank" will exactly cancel each other.

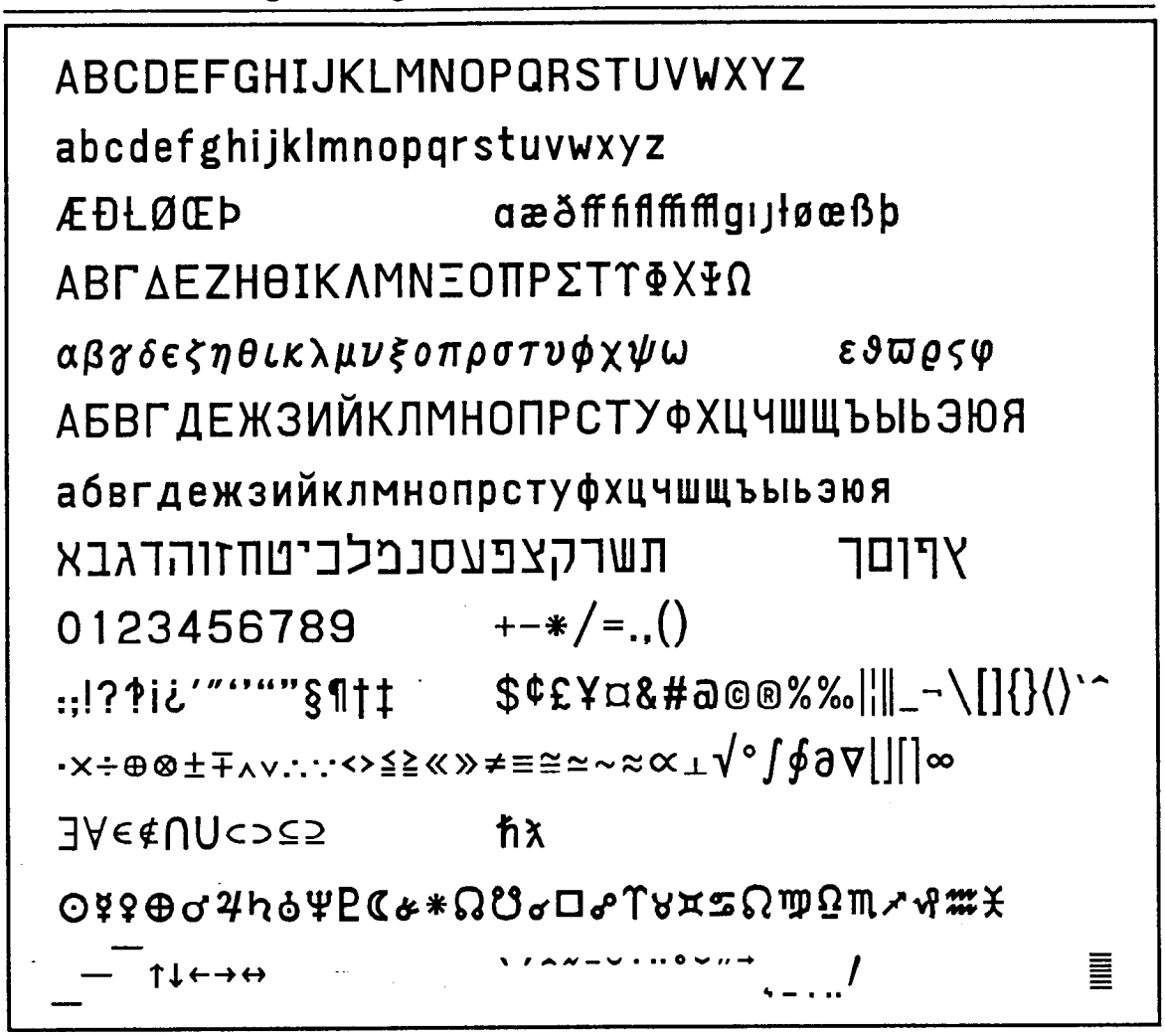

Figure 3.3. The solid font of the extended character set

The extended character generators usually produce characters of differing widths; thus the upper case letter "M" is about twice as wide as the upper case "I", and most lower case letters are about three-fourths as wide as most upper case letters. This results in a more pleasing appearance, but also causes some problems. If, for example, a letter is to carry both a superscript and subscript, something equivalent to a backspace would be necessary, but the amount backspaced would depend on the characters in the superscript (or subscript). To overcome this problem, a group of position control characters have been introduced which cause the stroke generator to save its current position and state. Another control character in a later part of the string can cause the earlier state of the stroke generator to be restored. There are four independent save-restore control character pairs available. The scope of these save-restore pairs is a single call to subroutine GZET. That is, you cannot save a position in one call to GZET and try to use it in a later call. If you try to use a position without saving it in an earlier part of the string, you will obtain the position of the beginning of the string.

3.1. Special Text Functions

This section describes a subroutine that gives the user control over the output of the character generator. Using this subroutine, it is possible for the user, for example, to produce projective transformations of the characters.

3.1.1. Subroutine GZETS: Extended Text Data

This subroutine may be used to process a string of characters in a manner similar to the way GZET does. However, instead of sending the data directly to the workstation, this subroutine calls a user supplied subroutine with the data. That subroutine can do anything it wants with the data.

The calling sequence is:

```
CALL GZETS(SUBR,PX,PY,PCHARS,SCHARS)
```

The input parameters are:

SUBR	An external variable specifying the subroutine to which the computed polylines or fill areas will be sent. The calling sequence of the subroutine is the same as subroutine GPL and GFA.
PX	A real value that gives the x coordinate of the location point of the character string in world coordinates.
PY	A real array that gives the y coordinate of the location point of the character string in world coordinates.
PCHARS	A character string containing the primary characters.
SCHARS	A character string containing the secondary characters.

3.1.2. GKS Implementation

Subroutine GZETS is very similar to subroutine GZET. The basic difference is that GZETS is simpler because it does not have to save and restore the current state of the polyline or fill area primitives; in this case that problem is up to the user.

Chapter 4

The GKS Subroutines and Enumeration Types

Both individuals and projects may outgrow GKS-EZ. This chapter contains a list of all of the GKS subroutines and enumeration types that are defined for FORTRAN-77. By referring to these lists, the user should be able to get some idea of the facilities of GKS that are not supported in GKS-EZ.

4.1. The GKS Subroutines

The following list contains the subroutine name, a short description of its function, and the level in which it first appears. A double asterisk after the level indicates that the subroutine is a user callable part of GKS-EZ; a single asterisk means that it is used internally in GKS-EZ. The organization and order of the list is the same as the GKS standards manuals *American National Standard for Information Systems: Computer Graphics - Graphical Kernel System (GKS) Functional Description* [ANS85a] and *American National Standard for Information Systems: Computer Graphics - Graphical Kernel System (GKS) FORTRAN Binding* [ANS85b].

Control Functions:

GOPKS	Open GKS	(ma)*
GCLKS	Close GKS	(ma)*
GOPWK	Open Workstation	(ma)*
GCLWK	Close Workstation	(ma)*
GACWK	Activate Workstation	(ma)**
GDAWK	Deactivate Workstation	(ma)**
GCLRWK	Clear Workstation	(ma)**
GRSGWK	Redraw All Segments on Workstation	(1a)
GUWK	Update Workstation	(ma)
GSDS	Set Deferral State	(1a)*
GMSG	Message	(1a)
GESC	Escape	(ma)

Output Functions:

GPL	Polyline	(ma)**
GPM	Polymarker	(ma)**
GTX	Text	(ma)**
GFA	Fill Area	(ma)**

GCA	Cell Array	(0a)
GGDP	Generalized Drawing Primitive	(ma)

Output Attributes:

Workstation Independent Primitive Attributes:

GSPLI	Set Polyline Index	(0a)
GSLN	Set Line Type	(ma)*
GSLWSC	Set Line Width Scale Factor	(0a)*
GSPLCI	Set Polyline Color Index	(ma)*
GSPMI	Set Polymarker Index	(0a)
GSMK	Set Marker Type	(ma)*
GSMKSC	Set Marker Size Scale Factor	(0a)*
GSPMCI	Set Polymarker Color Index	(ma)*
GSTXI	Set Text Index	(0a)
GSTXFP	Set Text Font and Precision	(0a)*
GSCHXP	Set Character Expansion Factor	(0a)
GSCHSP	Set Character Spacing	(0a)
GSTXCI	Set Text Color Index	(ma)*
GSCHH	Set Character Height	(ma)*
GSCHUP	Set Character Up Vector	(ma)*
GSTXP	Set Text Path	(0a)*
GSTXAL	Set Text Alignment	(ma)*
GSFAI	Set Fill Area Index	(0a)
GSFAIS	Set Fill Area Interior Style	(ma)*
GSFASI	Set Fill Area Style Index	(0a)*
GSFACI	Set Fill Area Color Index	(ma)*
GSPA	Set Pattern Size	(0a)*
GSPARF	Set Pattern Reference Point	(0a)*
GSASF	Set Aspect Source Flags	(0a)*
GSPKID	Set Pick Identifier	(1b)**

Workstation Attributes:

GSPLR	Set Polyline Representation	(1a)
GSPMR	Set Polymarker Representation	(1a)
GSTXR	Set Text Representation	(1a)
GSFAR	Set Fill Area Representation	(1a)
GSPAR	Set Pattern Representation	(1a)
GSCR	Set Color Representation	(ma)*

Transformation Functions:

Normalization Transformations:

GSWN	Set Window	(ma)*
GSVP	Set Viewport	(ma)*
GSVPIP	Set Viewport Input Priority	(mb)*
GSELNT	Select Normalization Transformation	(ma)**

Sample Input Functions:

GSMLC	Sample Locator	(mc)
GSMSK	Sample Stroke	(mc)
GSMVL	Sample Valuator	(mc)
GSMCH	Sample Choice	(mc)
GSMPK	Sample Pick	(1c)
GSMST	Sample String	(mc)

Event Input Functions:

GWAIT	Await Event	(mc)
GFLUSH	Flush Device Events	(mc)
GGTLC	Get Locator	(mc)
GGTSK	Get Stroke	(mc)
GGTVL	Get Valuator	(mc)
GGTCH	Get Choice	(mc)
GGTPK	Get Pick	(1c)
GGTST	Get String	(mc)

Meta File Functions:

GWITM	Write Item to GKS Meta File	(0a)
GGTITM	Get Item Type from GKS Meta File	(0a)
GRDITM	Read Item from GKS Meta File	(0a)
GIITM	Interpret Item	(0a)

Inquiry Functions:

Inquiry Functions for Operating State Value:

GQOPS	Inquire Operating State Value	(0a)*

Inquiry Functions for GKS Description Table:

GQLVKS	Inquire Level of GKS	(ma)
GQEWK	Inquire List (element) of Available Workstation Types	(0a)
GQWKM	Inquire Workstation Maximum Numbers	(1a)
GQMNTN	Inquire Maximum Normalization Transformation Number	(0a)*

Inquiry Functions for GKS State List:

GQOPWK	Inquire set (member) of Open Workstations	(0a)
GQACWK	Inquire set (member) of Active Workstations	(1a)
GQPLI	Inquire Polyline Index	(0a)
GQPMI	Inquire Polymarker Index	(0a)
GQTXI	Inquire Text Index	(0a)
GQCHH	Inquire Character Height	(ma)
GQCHUP	Inquire Character Up Vector	(ma)
GQCHW	Inquire Character Width	(0a)
GQCHB	Inquire Character Base Vector	(0a)
GQTXP	Inquire Text Path	(0a)
GQTXAL	Inquire Text Alignment	(ma)*

GQFAI	Inquire Fill Area Index	(0a)
GQPA	Inquire Pattern Size	(0a)
GQPARF	Inquire Pattern Reference Point	(0a)
GQPKID	Inquire Current Pick Identifier	(1a)
GQLN	Inquire Line Type	(ma)*
GQLWSC	Inquire Line Width Scale Factor	(0a)*
GQPLCI	Inquire Polyline Color Index	(ma)*
GQMK	Inquire Marker Type	(ma)
GQMKSC	Inquire Marker Size Scale Factor	(0a)
GQPMCI	Inquire Polymarker Color Index	(ma)
GQTXFP	Inquire Text Font and Precision	(0a)*
GQCHXP	Inquire Character Expansion Factor	(0a)
GQCHSP	Inquire Character Spacing	(0a)
GQTXCI	Inquire Text Color Index	(ma)
GQFAIS	Inquire Fill Area Interior Style	(ma)*
GQFASI	Inquire Fill Area Style Index	(0a)
GQFACI	Inquire Fill Area Color Index	(ma)*
GQASF	Inquire Aspect Source Flags	(0a)*
GQCNTN	Inquire Current Normalization Transformation Number	(ma)
GQENTN	Inquire List ⟨element⟩ of Normalization Transformation Numbers	(0a)
GQNT	Inquire Normalization Transformation	(ma)
GQCLIP	Inquire Clipping	(ma)
GQOPSG	Inquire Name of Open Segment	(1a)
GQSGUS	Inquire Set ⟨member⟩ of Segment Names in Use	(1a)
GQSIM	Inquire More Simultaneous Events	(mc)

Inquiry Functions for Workstation State List:

GQWKC	Inquire Workstation Connection and Type	(ma)
GQWKS	Inquire Workstation State	(0a)*
GQWKDU	Inquire Workstation Deferral and Update States	(0a)
GQEPLI	Inquire List ⟨element⟩ of Polyline Indices	(1a)
GQPLR	Inquire Polyline Representation	(1a)
GQEPMI	Inquire List ⟨element⟩ of Polymarker Indices	(1a)
GQPMR	Inquire Polymarker Representation	(1a)
GQETXI	Inquire List ⟨element⟩ of Text Indices	(1a)
GQTXR	Inquire Text Representation	(1a)
GQTXX	Inquire Text Extent	(ma)
GQEFAI	Inquire List ⟨element⟩ of Fill Area Indices	(1a)
GQFAR	Inquire Fill Area Representation	(1a)
GQEPAI	Inquire List ⟨element⟩ of Pattern Indices	(1a)
GQPAR	Inquire Pattern Representation	(1a)
GQECI	Inquire List ⟨element⟩ of Color Indices	(ma)
GQCR	Inquire Color Representation	(ma)
GQWKT	Inquire Workstation Transformation	(ma)

GQSGWK	Inquire Set ⟨member⟩ of Segment Names on Workstation	(1a)
GQLCS	Inquire Locator Device State	(mb)
GQSKS	Inquire Stroke Device State	(mb)
GQVLS	Inquire Valuator Device State	(mb)
GQCHS	Inquire Choice Device State	(mb)
GQPKS	Inquire Pick Device State	(1a)
GQSTS	Inquire String Device State	(mb)

Inquiry Functions for Workstation Description Table:

GQWKCA	Inquire Workstation Category	(0a)*
GQWKCL	Inquire Workstation Classification	(0a)
GQDSP	Inquire Display Space Size	(0a)*
GQDWKA	Inquire Dynamic Modification of Workstation Attributes	(1a)
GQDDS	Inquire Default Deferral State Values	(1a)
GQPLF	Inquire Polyline Facilities	(ma)
GQPPLR	Inquire Predefined Polyline Representations	(0a)
GQPMF	Inquire Polymarker Facilities	(ma)
GQPPMR	Inquire Predefined Polymarker Representation	(0a)
GQTXF	Inquire Text Facilities	(ma)
GQPTXR	Inquire Predefined Text Representation	(0a)
GQFAF	Inquire Fill Area Facilities	(ma)
GQPFAR	Inquire Predefined Fill Area Representation	(0a)
GQPAF	Inquire Pattern Facilities	(0a)
GQPPAR	Inquire Predefined Pattern Representation	(0a)
GQCF	Inquire Color Facilities	(ma)
GQPCR	Inquire Predefined Color Representation	(0a)
GQEGDP	Inquire List ⟨element⟩ of Available Generalized Drawing Primitives	(ma)
GQGDP	Inquire Generalized Drawing Primitive	(0a)
GQLWK	Inquire Maximum Length of Workstation State Tables	(0a)*
GQSGP	Inquire Number of Segment Priorities Supported	(1a)
GQDSGA	Inquire Dynamic Modification of Segment Attributes	(1a)
GQLI	Inquire Number of Available Logical Input Devices	(mb)*
GQDLC	Inquire Default Locator Device Data	(mb)
GQDSK	Inquire Default Stroke Device Data	(mb)
GQDVL	Inquire Default Valuator Device Data	(mb)
GQDCH	Inquire Default Choice Device Data	(mb)
GQDPK	Inquire Default Pick Device Data	(1b)
GQDST	Inquire Default String Device Data	(mb)

Inquiry Functions for Segment State List:

GQASWK	Inquire Set ⟨member⟩ of Associated Workstation	(1a)
GQSGA	Inquire Segment Attributes	(1a)

Inquiry Functions for Pixels:

GQPXAD	Inquire Pixel Array Dimensions	(0a)
GQPXA	Inquire Pixel Array	(0a)

GQPX Inquire Pixel . . . . . (0a)

Inquiry Functions for GKS Error State List:

GQIQOV Inquire Input Queue Overflow . . . . . (mc)

Utility Functions:

GEVTM Evaluate Transformation Matrix . . . . . (1a)

GACTM Accumulate Transformation Matrix . . . . . (1a)

Error Handling:

GECLKS Emergency Close GKS . . . . . (0a)*

GERHND Error Handling . . . . . (0a)

GERLOG Error Logging . . . . . (0a)

Utility Functions not Defined in GKS:

GPREC Pack Data Record . . . . . (0a)

GUREC Unpack Data Record . . . . . (0a)

GKS-EZ uses 17 of the GKS subroutines directly and another 56 subroutines indirectly. It therefore uses a total of 73 GKS subroutines. The total number of GKS subroutines for FORTRAN-77 in a Level 2c implementation is given by the following table:

Control Functions		12
Output Functions		6
Output Attributes		31
	Workstation Independent Primitive Attributes	25
	Workstation Attributes	6
Transformation Functions		7
	Normalization Transformations	5
	Workstation Transformation	2
Segment Functions		13
	Segment Manipulation Functions	8
	Segment Attributes	5
Input Functions		32
	Initialization of Input Devices	6
	Setting Mode of Input Devices	6
	Request Input Functions	6
	Sample Input Functions	6
	Event Input Functions	8
Metafile Functions		4
Inquiry Functions		100
	Inquiry Functions for Operating State Value	1
	Inquiry Functions for GKS Description Table	4
	Inquiry Functions for GKS State List	36

Inquiry Functions for Workstation State List	24
Inquiry Functions for Workstation Description Table	29
Inquiry Functions for Segment State List	2
Inquiry Functions for Pixels	3
Inquiry Functions for GKS Error State List	1
Utility Functions	2
Error Handling	3
Utility Functions not Defined in GKS	2
Total	212

4.2. The GKS Enumeration Types

Many of the GKS subroutines contain input or output parameters that assume a small number of integer values. GKS assigns enumeration types to these values so that a programmer may refer to them symbolically in a consistent manner. A full list of the enumeration types follows:

Aspect Source:
- `GBUNDL = 0` Bundled
- `GINDIV = 1` Individual

Clear Control Flag:
- `GCONDI = 0` Clear display if not empty
- `GALWAY = 1` Clear display always

Clipping Indicator:
- `GNCLIP = 0` No clipping
- `GCLIP = 1` Clip

Color Available:
- `GMONOC = 0` Monochrome display
- `GCOLOR = 1` Color display

Coordinate Switch:
- `GWC = 0` World coordinates (WC)
- `GNDC = 1` Normalized device coordinates (NDC)

Deferral Mode:
- `GASAP = 0` As soon as possible
- `GBNIG = 1` Before next interaction globally
- `GBNIL = 2` Before next interaction locally
- `GASTI = 3` At some time

Detectability:
- `GUNDET = 0` Undetectable
- `GDETEC = 1` Detectable

Device Coordinate Units:
- `GMETRE = 0` Meters
- `GOTHU = 1` Other

Display Surface Empty:

```
   GNEMPT = 0    Not Empty
   GEMPTY = 1    Empty
Dynamic Modification:
   GIRG   = 0    Implicit Regeneration
   GIMM   = 1    Immediately
Echo Switch:
   GNECHO = 0    No Echo
   GECHO  = 1    Echo
Fill Area Interior Style:
   GHOLLO = 0    Hollow
   GSOLID = 1    Solid
   GPATTR = 2    Pattern
   GHATCH = 3    Cross hatched
Highlighting:
   GNORML = 0    Normal
   GHILIT = 1    Highlighted
Input Device Status:
   GNONE  = 0    None
   GOK    = 1    OK
   GNPICK = 2    No Pick
   GNCHOI = 2    No Choice
Input Class:
   GNCLAS = 0    None
   GLOCAT = 1    Locator
   GSTROK = 2    Stroke
   GVALUA = 3    Valuator
   GCHOIC = 4    Choice
   GPICK  = 5    Pick
   GSTRIN = 6    String
Implicit Regeneration Mode:
   GSUPPD = 0    Suppressed
   GALLOW = 1    Allowed
Level of GKS:
   GLMA   =-3    Level ma
   GLMB   =-2    Level mb
   GLMC   =-1    Level mc
   GL0A   = 0    Level 0a
   GL0B   = 1    Level 0b
   GL0C   = 2    Level 0c
   GL1A   = 3    Level 1a
   GL1B   = 4    Level 1b
   GL1C   = 5    Level 1c
   GL2A   = 6    Level 2a
   GL2B   = 7    Level 2b
```

GL2C = 8 Level 2c

New Frame Action Necessary:

GNO = 0 No
GYES = 1 Yes

Operating Mode:

GREQU = 0 Request
GSAMPL = 1 Sample
GEVENT = 2 Event

Operating State Value:

GGKCL = 0 GKS closed
GGKOP = 1 GKS open
GWSOP = 2 At least one workstation open
GWSAC = 3 At least one workstation active
GSGOP = 4 Segment open

Presence of Invalid Values:

GABSNT = 0 Absent
GPRSNT = 1 Present

Regeneration Flags:

GPOSTP = 0 Postpone
GPERFO = 1 Perform

Relative Input Priority:

GHIGHR = 0 Higher
GLOWER = 1 Lower

Simultaneous Events Flag:

GNMORE = 0 No more
GMORE = 1 More

Text Alignment Horizontal:

GAHNOR = 0 Normal
GALEFT = 1 Left
GACENT = 2 Center
GARITE = 3 Right

Text Alignment Vertical:

GAVNOR = 0 Normal
GATOP = 1 Top
GACAP = 2 Cap
GAHALF = 3 Half
GABASE = 4 Base
GABOTT = 5 Bottom

Text Path:

GRIGHT = 0 Right
GLEFT = 1 Left
GUP = 2 Up
GDOWN = 3 Down

Text Precision:

```
   GSTRP  = 0   String
   GCHARP = 1   Character
   GSTRKP = 2   Stroke
Type of Returned Values:
   GSET   = 0   Set
   GREALI = 1   Realized
Update State:
   GNPEND = 0   Not pending
   GPEND  = 1   Pending
Vector/Raster/Other Type:
   GVECTR = 0   Vector
   GRASTR = 1   Raster
   GOTHWK = 2   Other
Visibility:
   GINVIS = 0   Invisible
   GVISI  = 1   Visible
Workstation Category:
   GOUTPT = 0   Output
   GINPUT = 1   Input
   GOUTIN = 2   Output and Input
   GWISS  = 3   Workstation Independent Segment Storage
   GMO    = 4   GKS Meta File Output
   GMI    = 5   GKS Meta File Input
Workstation State:
   GINACT = 0   Inactive
   GACTIV = 1   Active
List of Generalized Drawing Primitive Attributes:
   GPLATT = 0   Polyline Attribute
   GPMATT = 1   Polymarker attribute
   GTXATT = 2   Text Attribute
   GFAATT = 3   Fill Area Attribute
Line Type:
   GLSOLI = 1   Solid
   GLDASH = 2   Dashed
   GLDOT  = 3   Dotted
   GLDASD = 4   Dashed-Dotted
Marker Type:
   GPOINT = 1   Point "."
   GPLUS  = 2   Plus "+"
   GAST   = 3   Asterisk "*"
   GOMARK = 4   Circle "O"
   GXMARK = 5   Cross "X"
Attribute Control Flag:
   GCURNT = 0   Current
```

```
   GSPEC  = 1    Specified
```
Polyline/Fill Area Control Flag:
```
   GPLINE = 0    Polyline
   GFILLA = 1    Fill Area
```
Initial Choice Prompt Flag:
```
   GPROFF = 0    Off
   GPRON  = 1    On
```

GKS therefore defines a total of 126 enumeration types.

References

The following list contains more information about the books and reports that have been referenced in this document.

[ANS78] *American National Standard: Programming Language FORTRAN*, Document ANSI X3.9–1978, American National Standards Institute, Inc., New York, April 1978.

[ANS85a] *American National Standard for Information Systems: Computer Graphics – Graphical Kernel System (GKS) Functional Description*, Document ANSI X3.124–1985, American National Standards Institute, Inc., New York, June 1985.

[ANS85b] *American National Standard for Information Systems: Computer Graphics – Graphical Kernel System (GKS) FORTRAN Binding*, Document ANSI X3.124.1–1985, American National Standards Institute, Inc., New York, June 1985.

[ANS86] *American National Standard for Information Systems: Coded Character Sets, 7-bit American National Standard Code for Information Interchange (7-bit ASCII)*, Document ANSI X3.4–1986, American National Standards Institute, Inc., New York, March 1986.

[Bar66] J. E. Barmack and H. W. Sinaiko, *Human Factors Problems in Computer-Generated Graphic Displays*, Study S-234, Institute for Defense Analysis, Research and Engineering Support Division, April 1966.

[Fol74] J. D. Foley and V. L. Wallace, "The art of natural graphic man-machine conversation," *Proceedings of the IEEE*, Volume 62, Number 4, pages 462–471, April 1974.

[Her67] A. V. Hershey, *Calligraphy for Computers*, Report Number 2101, United States Naval Weapons Laboratory, Dahlgren, Virginia, August 1967.