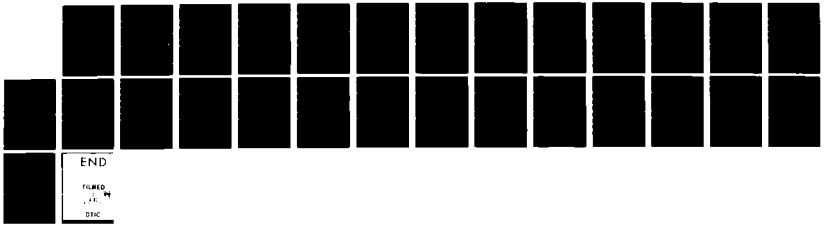
AD-A126 603 C2 GRAPHICS EDITOR USER'S MANUAL(U) UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST D HOLLINGWORTH FEB 83 ISI/TM-83-24 1/1

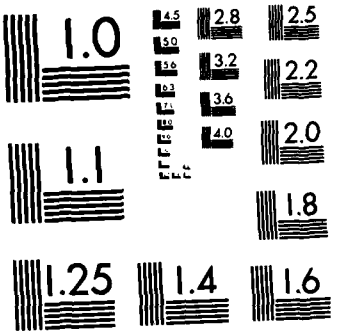
UNCLASSIFIED MDA903-81-C-0335 F/G 9/2 NL



END

FILMED

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ISI/TM-83-24
February 1983

C2 Graphics Editor User's Manual

Dennis Hollingworth

INFORMATION SCIENCES INSTITUTE
UNIVERSITY OF SOUTHERN CALIFORNIA
4676 Admiralty Way/Marina del Rey/California 90291
(213) 822-1511

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER ISI/TM-83-24	2. GOVT ACCESSION NO. AD-A126 603	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* C2 Graphics Editor User's Manual		5. TYPE OF REPORT & PERIOD COVERED Technical Manual
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Dennis Hollingworth		8. CONTRACT OR GRANT NUMBER*(s)* MDA903 81 C 0335
9. PERFORMING ORGANIZATION NAME AND ADDRESS USC/Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90291		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE February 1983 13. NUMBER OF PAGES 28
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

This document is approved for public release and sale; distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

.....

18. SUPPLEMENTARY NOTES

.....

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

command and control graphics, computer graphics, device-independent graphics system, graphics editor, graphics files, high-level graphics language, on-line map display

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

(OVER)

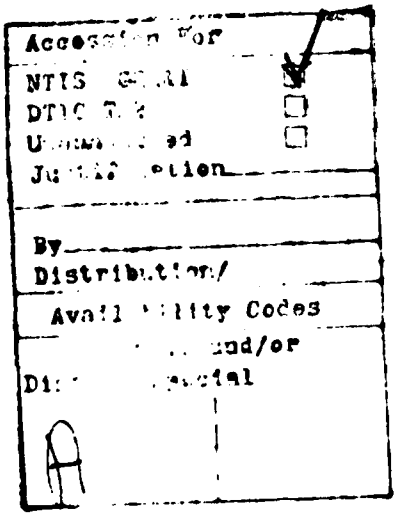
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. ABSTRACT

The C2 Graphics Editor is a Graphics Language (GL) application program intended to facilitate the creation of graphics files for use by the Briefing Aid and other GL application programs. These graphics files are standard GL device-independent files and may be played back by any GL application program on any GL-supported graphics output device. The editor will run on both TENEX and TOPS-20 machines on which GL is available. The graphics editor consists of an application component linked with a GL frontend, which is bound at run-time to a GL backend. As with any GL application program, it executes either on a single machine or on multiple machines with an ARPANET connection between the application/frontend and the graphics backend.



ISI/TM-83-24
February 1983

C2 Graphics Editor User's Manual

Dennis Hollingworth

INFORMATION SCIENCES INSTITUTE

UNIVERSITY OF SOUTHERN CALIFORNIA

4676 Admiralty Way/Marina del Rey/California 90291
(213) 822-1511

This research is supported by the Defense Advanced Research Projects Agency under Contract No. MDA903 81 C 0335. Views and conclusions contained in this report are the author's and should not be interpreted as representing the official opinion or policy of DARPA, the U S Government, or any person or agency connected with them.

CONTENTS

FUNCTIONAL CHARACTERISTICS

INTRODUCTION

The C2 Graphics Editor is a Graphics Language (GL [Bisbey 80]) application program intended to facilitate the creation of graphics files for use by the Briefing Aid and other GL application programs. These graphics files are standard GL device-independent files and may be played back by any GL application program on any GL-supported graphics output device. The editor will run on both TENEX and TOPS-20 machines on which GL is available. The graphics editor consists of an application component linked with a GL frontend, which is bound at run-time to a GL backend (Figure 1). As with any GL application program, it executes either on a single machine or on multiple machines with an ARPANET connection between the application/frontend and the graphics backend.

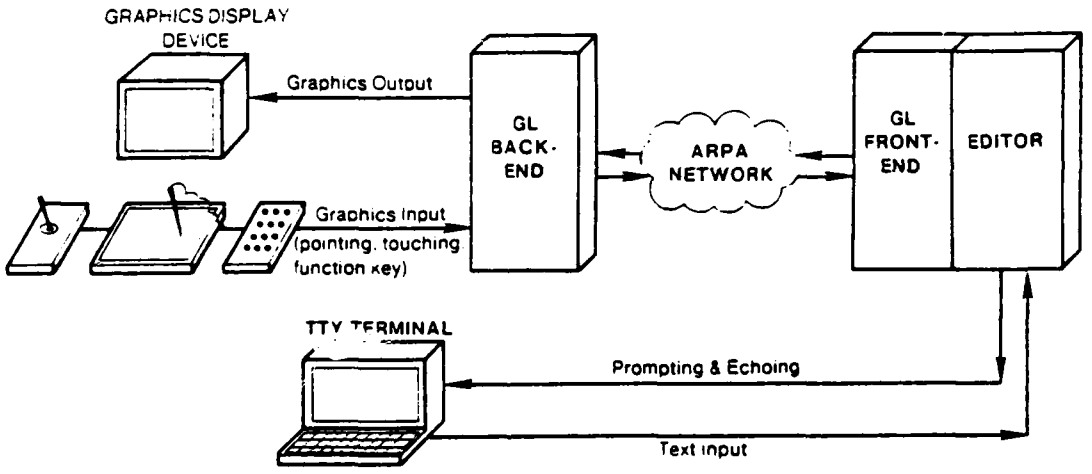


Figure 1

GRAPHICS EDITOR START-UP

The graphics editor exists as an application program in the <LEVEL2> directory of various ISI machines. It utilizes two terminals, a TTY terminal for text input and a graphics terminal for picture display and graphics input. Various forms of required input may come from either of these terminals. The editor is started like any other application program. The user proceeds through an interactive

dialogue to determine the backend location, the backend type, and the device location, similar to that employed for the Briefing Aid system. In addition, the editor queries the user as to the terminal to be used for TTY input during graphics editor operation. (See the appendix for an example of the start-up dialogue.)

Once started, the editor creates a box around an editor work area on the graphics output display device and generates a command menu to the right of the work area. The user typically touch-selects commands and their associated subcommands via menu element selection in response to menu prompting, pointing to objects or locations on the display surface or entering numerical data or text at the TTY as required. The user then creates as many individual GL files as he desires, progressively building up pictures or overlays until he has completed the session. When he is finished, the user terminates editor operation, returning control to the operating system.

GRAPHICS EDITOR INPUT

The graphics editor creates and manipulates graphics editor objects in response to user-entered commands. A user-entered command consists of a sequence of options (subcommands), typed or selected in order, along with specified numerical data values or screen coordinates which either select a specific mode of operation or describe an operation to be performed. At any point during command entry, the user may back up to preceding points in the command specification sequence (e.g., when an error has been made) and respecify succeeding elements of the command. Possible elements of an editor command include subcommand, location, and naming input and may be entered through either of two input sources, a TTY terminal for ASCII text input, or a graphics device with pointing, touching and function key input. Through these commands the user creates graphics files made up of colored or highlighted lines, text, icons, and filled areas, as well as existing graphics files.

While the graphics editor is intended to simultaneously support multiple input sources for individual command elements, its current implementation requires that a particular input type be associated with a specific input source during entry of a complete editor command. This association may be changed by the user, however. Thus, subcommand input, which indicates desired modes of operation of a full editor command, is initially accomplished via menu item selection but, alternatively, may be entered via the user's TTY. Location input, used to indicate a specific location on the display screen, is initially supplied via the pointing facility of the graphics device, but may be supplied by entering coordinates' values on the TTY device. Naming input, used to indicate particular editor objects to be acted upon, is initially supplied by the touching mechanism on the graphics device but may be supplied by entering an object's name on the user's TTY device. Selecting the current source of each type of user input is accomplished through the ACCEPT command.

SUBCOMMAND ENTRY

During specification of a command, the user supplies a series of subcommands and data values indicating the various options and screen coordinates desired. When working at the TTY, the user

terminates a given subcommand by typing a space, tab, or carriage return. When working through the graphics device, the user selects the desired subcommand from a menu by placing the graphics cursor on the appropriate character string and striking the function key "0".

ABBREVIATED COMMAND INPUT

It is not necessary for the user to type textual subcommands in their entirety. Two abbreviated modes of command input are supported by the graphics editor. The user may type any sequence of characters which uniquely identifies the command desired and then terminate the command with a space, tab, or carriage return as indicated above. If this string is sufficient to specify the command desired, the graphics editor will accept the abbreviated command string. Thus, the characters "CO" might be typed for the command COLOR, while the character "A" might be sufficient to specify the subcommand ACCEPT. If the command is not yet unique, the terminal *beeps* and waits for additional character string input.

ESC KEY

If the user enters an abbreviated command but desires to see the entire subcommand displayed on the TTY device, he may terminate the partial string with the escape key. The escape key requests that the graphics system complete the remainder of the subcommand. As above, if the string is not yet unique, the terminal *beeps* and waits for additional character string input.

UNDOING/BACKING UP

In the process of entering a command the user may make a mistake. Until a command is completed, mistakes may be undone or backed up over both on the TTY device and on the graphics device. Backing up on the TTY device involves striking the backspace, control-A, or delete keys. If the user is in the middle of typing a string of characters for a subcommand, striking one of these keys causes the cursor to back up one character. If the user backs up beyond the first character of the subcommand, the cursor is moved to the start of the previous subcommand or data argument, requiring that the user reenter the item. The user may back all the way out of a command, in which case a new command prompt will appear.

It is also possible to back up on the graphics display device. Whenever graphic input is expected, whether it's subcommand, pointing, or touching input, a string of left arrows (<<<<) appears at the upper right corner of the display surface. Touch-selecting the string results in the latest subcommand selection, touching action, or pointing operation being voided. The appropriate menu selection list is generated at the right of the display area as preceding command levels are reentered.

Certain GL commands require specification of a sequence of characters, vectors, or data points. Like subcommands, these may also be backed up over. The most recent item in a vector sequence may

be voided by touching the string of left arrows on the graphics display device (and then the next most recent, and the next most recent, and so on). Similiarly, successive text characters may be deleted by repeated backspacing on the TTY device during text string input.

To minimize screen erasures, the effects of such action are not reflected on the display surface until a new data value or text character is entered unless the user has backed completely out of the data entry subcommand mode (i.e., he has deleted all data or text items entered for this command level). When a new data value is entered, the graphic display is refreshed to reflect the current state of the command.

GRAPHICS EDITOR OUTPUT

Graphics editor output takes one of several forms. It may take the form of prompting, of command entry or error feedback, or it may reflect the current state of the graphic file under development. In the case of prompting, the output may appear as a text prompt on the TTY device or as a menu or cursor on the graphics display device. In the case of command entry feedback, most output is directed to the user's TTY device. In the case of error feedback, the output again usually appears on the TTY. In the case of file status feedback, the output results in a picture in the editor work area on the display device.

PROMPTING

A user prompt is supplied for many of the command elements accepted from the TTY or graphics device. On the TTY device this applies primarily to expected data values, such as the center point for an arc, and takes the form of a textual prompt. On the graphics display device, a cursor appears on the screen whenever graphic input is expected and if command input is expected, a command menu is established to the right of the graphic working area.

COMMAND ENTRY FEEDBACK

As a subcommand or object is selected or a location is entered via the display device, the action is echoed on the TTY device. This provides feedback to the user as well as a transcript of his editor interaction.

ERROR FEEDBACK

Errors may be detected by the editor during the user's entry of a subcommand or a data value at the

TTY device. When an error is detected the terminal *beeps* and the cursor is repositioned at the start of the current command string or data value. The user is expected to retype the subcommand or data value that was in error and proceed with the remainder of the operation.

GRAPHICS EDITOR OBJECTS

The graphics editor creates and operates on graphics editor objects. A graphics editor object is the result of one or more editor operations by the user. It is a collection of text, lines, and filled areas that are logically associated with one another. Any graphics editor command affects the entire object selected. (For example, if a single character which is part of a more complex object is selected during a **MOVE** operation, the entire object will be moved to the new location.)

Individual pieces of an editor object are created via the **CREATE** command, but may be manipulated via other editor commands. Separately created objects may be joined together by the **JOIN** command, thereby making a composite object or, broken up via the **BREAK** command, creating multiple objects, which then require individual manipulation in subsequent operations.

The individual pieces of an editor object have an implicit order associated with them which affects command interpretation. Individual character strings which have been created are ordered from the first to the last character of the string; individual vector sequences are ordered from the first to the last vector entered. If two objects are joined, one object's elements will logically precede the other object's elements in the composite editor object. This has particular significance when the user attempts to change sections of an editor object. For example, as part of the **CHANGE COLOR** operation the user specifies the starting and ending elements to be involved. If the creation order of the editor object is not known, or is not followed, an unanticipated change may occur, e.g., everything other than the desired range.

DATA SEPARATION (SCOPES)

The graphics editor allows the user to separate sets of data from other sets of data on a logical basis. The editor supports fifteen logical scopes into which individual files may be read or with which particular graphic objects may be associated. While generating a graphics file, the user specifies the scopes to be copied into the file. This allows him to create overlay information on top of existing graphics file data, saving only the overlay information. Likewise, it allows him to aggregate editor objects, writing them out as separate overlay files.

Logical scopes do not necessarily correspond to the physical scopes in the graphics system. However, the editor does request the graphics system scope specified by the user in the logical scope request. If the particular backend that is linked to the frontend supports multiple scopes, the capability may be exploited by the graphics editor user using this technique.

WORK GRID

A work grid is associated with the graphics editor work area at all times during editor operation although it may or may not be visible at any given moment depending upon the user's desires. The spacing between individual grid points is determined via the **SET GRID SIZE** command. The grid is used for several purposes. It determines the size of the characters which will be created by the **CREATE TEXT** command of the graphics editor. Each grid section corresponds to the height and width of the selected character set envelope. (See the section on text fonts in the GL manual [Bisbey 80]).

The grid is also utilized in certain modes of line creation. In both the right and normalized forms of linked line creation, the terminating point of a given link is determined by either the nearest grid point or the nearest right grid point to the terminus of the current line.

Grid size may be changed from extremely large to extremely small inter-grid element spacing and the grid itself may be either visible or invisible at any given moment of graphics editor operation, depending upon the choice of the user.

GRAPHICS EDITOR COMMANDS

The following commands are recognized by the current graphics editor implementation. Acceptable sources of input for individual command arguments vary with each command. Location or pointing input might be entered at the I/O device or on the TTY device depending upon the input mode that was selected for location input. Some types of input are only accepted at a single device, no matter what the state of the global input type as specified by the user. For example, certain types of pointing input may only be accepted from the graphics input device, whereas character string input is only accepted from the TTY device. Prompting indicates where input should be generated for the current input mode settings.

CHANGING DEFAULT INPUT MODES

When the graphics editor is first started, various default input mechanisms are automatically established for the user. The **ACCEPT** command allows the user to change these to fit his own individual preferences. Editor commands may be entered either from the TTY or by menu selection on the graphics device; locations may be entered from the TTY or via the pointing device; object names may be entered from the TTY or via the touching mechanism on the graphics device. During an editor session, the user may switch between the various input source options as frequently as he likes.

ACCEPT {LOCATIONS, COMMANDS, NAMES} from {TTY, DEVICE}

The user selects the input mechanism to be used for a particular form of input via the **ACCEPT** command. Input may be from either the TTY or the graphics device. This command has certain side-effects on user/editor interaction. If command input is entered at the TTY device, then no menu appears on the graphics display device. If, on the other hand, command input is accepted from the graphics display device, a menu appears to the right of the graphics editor work area. If location input is entered via the TTY device, the user types (in response to prompts which appear on the TTY) the desired X and Y coordinates within the coordinate range established for the graphics editor work area. If location input is accepted from the graphics device, the user specifies the desired X and Y coordinates by utilizing the pointing mechanism to indicate a location within the graphics editor work area.

After entering the accept command, the user first selects the type of input with which he is concerned (i.e., location input, command input), and then, in response to a subsequent prompt, the source of this input (i.e., TTY or graphics device). Subsequent input of the indicated type will then be accepted only from the device indicated until the user once again changes the input source.

ESTABLISHING OBJECT DISPLAY ATTRIBUTES

Default object display attributes are established by the graphics editor at start-up and apply to all objects subsequently created. Thus, lines and text are created under the color and highlighting attributes in effect at the time of object creation. These default values may be changed by the user via the **SET** command.

SET { GRID, COLOR, LINETYPE, FILLING, STYLE, HIGHLIGHT, FILEAREA }

The **SET** command is used to change display mode parameters as well as establish default values for the display characteristics of graphic objects subsequently defined by the **CREATE** command.

SET GRID { SIZE, DISPLAY }

The user controls selected display grid characteristics via the **SET GRID** option of the set command. In response to the **SET GRID** command, he specifies either a grid size to be established or the visibility attribute of that grid.

SET GRID SIZE to *(integer)*

Grid size values range from 1 to 20 and correspond to specific text font sizes in increments of 4 coordinate units wide by 5 units high. The starting text size is a 4 by 5 box in the 1024 square coordinate system initially applied to the editor work area. Successive grid sizes then increase this box by 4 units in the horizontal direction and 5 units in the vertical direction. The default grid size value is **9**.

SET GRID DISPLAY to { ON, OFF }

Visibility or invisibility of the current grid may be controlled by the **SET GRID DISPLAY** command. Grid visibility may be either **ON** or **OFF**. If grid visibility is **ON**, a dot grid appears on the graphics editor work area. An adjacent set of four dots indicates the boundaries of a character envelope. These grid points may also be used to control the orientation and terminus of vectors created via the **CREATE LINES** command. The default grid display status is **OFF**.

SET COLOR to *(saturation) (hue)*

The COLOR subcommand allows the user to specify the default color in which graphic primitives subsequently defined will be drawn. Having selected the SET COLOR operation, the user specifies the desired saturation (PALE, MEDIUM, or VIVID) for any one of the supported hues. These include RED, YELLOW, GREEN, BLUE, CYAN, MAGENTA, and WHITE. Subsequently created graphic objects will appear in the default color setting until that value is changed. (If the currently connected backend does not support the indicated color it will, be mapped at display time to the nearest available color. The file produced will be written with the requested color specification however.) The system start-up default color setting is VIVID WHITE.

SET LINETYPE to { DIRECT, NORMALIZE, RIGHT } { PLAIN, ARROWED }

The LINETYPE option of the SET command allows the user to specify default line characteristics for lines subsequently created via a CREATE command. The first set of defaults determines how terminal points of the entered line segment relate to the grid. The user may specify that line segments be DIRECT -- that they start and terminate exactly at the locations specified. He may request that line terminus values be NORMALIZED -- that they be adjusted to the nearest grid point. The user may request that all lines subsequently created be RIGHT lines -- adjusted to either a vertical or horizontal orientation (whichever is closest to the actual direction specified).

The user next specifies how the indicated line segment terminates, i.e., whether or not it terminates with an arrowhead. The system start-up default linetype setting is DIRECT PLAIN.

SET FILLING to { NOFILL, SINGLE, DOUBLE, SOLID }

The FILLING subcommand allows the user to determine the default filling characteristics for objects after they've been created. After specifying the FILLING subcommand, the user may choose between NOFILLing, SINGLE-hatched filling, DOUBLE-hatched filling, or SOLID filling. The system start-up default is SINGLE-hatched filling.

SET STYLE to { UPPER,LOWER,UPLOW } { ASCII,CYRILLIC,MATH,ICON,BLOCK,CAPS }

The STYLE subcommand allows the user to specify the text style of characters subsequently entered via a CREATE TEXT command. The user first establishes whether characters should be translated to uppercase, (2) translated to lowercase, or (3) left as they were entered in upper-lowercase mode. Having selected the desired translation, the user next specifies the desired text font to be used for character generation. Supported text fonts are the same as described in the *Graphics Language Manual* and include ASCII, CYRILLIC, MATH, ICON, BLOCK LETTER, and CAPS character sets. The default style setting is UPLOW ASCII.

SET HIGHLIGHT to { ON, OFF }

The **HIGHLIGHT** subcommand allows the user to control whether subsequently created graphic objects will be highlighted or unhighlighted as an object creation default characteristic. Initially, all objects are created with highlighting **OFF**.

SET FILEAREA from *(x,y pair)* to *(x,y pair)*

The **FILEAREA** subcommand allows the user to specify a window within the graphics editor work area in which GL files will be played back. The user specifies the window by indicating the two opposite corner vertices of a play-back box. Initially, the play-back window is set to the editor work area.

SET RANGE along { X-AXIS, Y-AXIS } from: *(x,y pair)* to *(x,y pair)*
using coordinate range from: *(number)* to *(number)*

The **RANGE** subcommand allows the user to specify a user-to-system coordinate mapping process. It allows the user to subsequently enter coordinates in his own coordinate system when accepting location input from the TTY device. The user first indicates two points along the X or Y axis onto which his own coordinate system points will be mapped. He then specifies two values in the desired coordinate range which will correspond to the indicated points. A linear mapping is established between the desired coordinate range and the indicated display location points. Subsequent screen location data must be entered in the newly selected coordinate range. All user feedback is subsequently supplied in terms of the new coordinate range. Thus, the user might select two points on the X-axis of a currently displayed graph and indicate that those points will correspond to a specific coordinate range. He could then cause a desired point plot to appear in the indicated area on the display screen. Initially, the X and Y axes are set to 0.0 - 1023.0.

CONTROLLING OVERLAY SEPARATION

In many cases the user wishes to create several graphics files which are related to one another or overlay one another, while keeping the files physically separate. He does this by utilizing a logical overlay mechanism defined for the graphics editor which allows him to isolate one set of information from another. The individual overlays can subsequently be written onto graphic files and then used collectively or individually. Logical overlay selection is controlled via the **SELECT** command.

SELECT SCOPE NUMBER: *(number)*

The **SELECT** command determines the logical scope into which GL files will be read and newly created editor objects will be established. The data in each scope is logically separate from that in other scopes, enabling the user to do such things as create overlay information on top of a background map (where the background map resides in one scope and the overlay information in another). In response to a scope number prompt, the user enters the desired scope number on the TTY. Only one scope may be active at a time, with the default scope number being scope number 3. (Scopes 1 and 2 are used for the grid and command prompt information, although they may also be selected via this command. In this case some display update degradation will occur when a multi-scope backend is selected).

While the scope numbers specifiable by this command are only logical scope numbers, intended to allow the user to partition data according to his specific needs, they may correspond to physical scopes supported by the connected graphics system backend. In particular, the graphics editor will attempt to select the GL scope which corresponds in number to the logical scope selected via this command. If there is no corresponding physical scope, the graphics system maps the specified scope into its current default scope value.

CREATING AN OBJECT

A graphics editor object consists of a logically related set of lines, texts, arcs, sectors, or filled areas which are treated by the editor as a single entity. The individual elements comprising an object are specified via the **CREATE** command. After it has been created, sections of a non-textual object may be filled via the **FILL** command. These two commands are described below. Two or more objects may be combined to create more complex objects. These and other operations which may be performed on an object are described in subsequent sections of this document.

CREATE { TEXT, LINES, ARCS, ARC, SECTOR }

The **CREATE** command allows the user to define graphics editor objects on the graphics editor work area. All user input received as a consequence of a single create operation becomes associated with a single graphics editor object. Other graphics editor commands manipulate this object in its entirety unless the object is broken up into smaller objects or associated with other graphics editor objects via the **BREAK** or **JOIN** editor operations.

The **CREATE** command has several submodes depending upon the type of graphics primitives to be created. They are as follows:

CREATE TEXT at ***(x,y pair)*** **"** ***(string)*** **"**

The text submode allows the user to enter an arbitrary sequence of text characters on the TTY device starting at a user-selected location in the graphics editor work area. After the text mode is selected, the user is prompted for the X,Y origin of the text string to be established. The text string representation depends upon the current font style that has been selected for graphics editor input as well as the grid size that is currently in effect.

The characters drawn on the display surface will appear in the current color selection and current highlighting modes. The text character envelope in which the character subsequently appears exactly matches the current grid size characteristics. Hence, the actual character location will be some place in between the boundaries of a given grid element and will be somewhat smaller than the actual grid size.

After the starting X,Y location has been specified, the user is prompted via a double quote prompt on the TTY device for the characters he wishes to enter. The user may then enter a character string on the TTY. The characters are echoed on the graphic display device as they are typed in. A carriage return included in the text string causes the next character to be positioned at the same X-location as the first character of the string, and at a Y-location one grid unit below. A double quote terminates text character input unless it is preceded by an escape key, in which case it is included as part of the text string.

If the user makes an error while entering a text string he may back up over erroneous characters by repeatedly striking the backspace, delete, or control-A key. This back up operation causes the cursor on the user's TTY to back up over characters in the text string, but has no immediate effect on the graphic display surface. When the user has finished backing up and types a new character, the graphic display surface is updated to reflect the altered text string.

The user may back up past a carriage return operation if he desires. In this case, the cursor is positioned immediately following the last character on the previous text line. Any new character which is input at this point will further extend that line of text.

CREATE LINES from x,y loc: ***(x,y pair)***
to x,y loc: ***(x,y pair)***
to x,y loc: ***(x,y pair)***
.
.
.

The user may create a series of linked line segments using the **CREATE LINES** option of the create command. Upon entering linked line creation submode, the user is immediately prompted for an originating X,Y location on the display device. Having specified that location, the user is then prompted for successive line segments with each succeeding line segment originating at the previous line segment's terminus and extending to the newly entered X,Y location. As the user creates the

linked line segment chain, each line segment which he has created is echoed on the work area of the graphics editor. Furthermore, the data points that the user has indicated via the input device are echoed on the user's TTY. These data values are represented in the user's coordinate range, extended to cover the entire graphics editor working area.

Should the user make a mistake during entry of consecutive line segments, he may back up over these line segments by selecting the backspace operation on the graphics editor display device. This is reflected on the TTY device by a prompt for a new X,Y coordinate, but has no immediate effect upon the graphics display device picture. The user continues to back up until he has deleted the segments he wanted to eliminate, at which point he enters a new terminus for the most recently deleted line segment. The picture on the graphics display device is immediately updated to reflect the current presentation of the linked line set being defined. The user terminates the **CREATE LINES** command by either typing a function key on the graphics display device or entering a carriage return in response to a coordinate prompt.

CREATE ARCS from: ***(x,y pair)*** **thru x,y loc:** ***(x,y pair)*** **to x,y loc:** ***(x,y pair)***
thru x,y loc: ***(x,y pair)*** **to x,y loc:** ***(x,y pair)***
.
.
.

The **CREATE ARCS** subcommand is used to create a series of linked arcs on the graphic output area which are treated by the editor as a single graphics editor object. Arcs are established by three points. As with successive line segments, the terminus of one arc functions as the starting point of the next arc.

Having specified the **CREATE ARCS** subcommand, the user is prompted for the starting X,Y location for the arc sequence. The user is next prompted for a second point on the arc, and then a terminus point for the curve. The user may terminate arc specification at this point or any such succeeding arc terminus point by striking a function key (other than 0) if input is from the graphics device, or a carriage return if input is expected from the TTY.

As successive points are entered, they are echoed on the graphics editor work area as a mark on the screen. When the three points which describe an arc have been entered, they are echoed as a completed curve. As successive sets of points are entered, a figure is created on the graphics work area consisting of a series of linked arcs of concave and/or convex shape. The user is subsequently prompted for successive mid-points and terminus points for new arc segments.

Like other **CREATE** subcommands, the user may back up over the entered data values by using the backspace operation on the current input source. The figure is changed to reflect the current data values when a new data point is entered.

CREATE ARC with center at: *(x,y pair)* **from** *(x,y pair)* **thru** *(x,y pair)* **to** *(x,y pair)*

As an alternative to the above technique, the user may create a single arc by utilizing a center, starting angle, radian point, and ending angle command sequence. When the user has specified the ARC submode, he is prompted for a center point for the arc to be created. Having selected a center point (echoed as a small cross on the editor work area), he is then prompted for the arc starting angle. The center point and this starting point define a radial line emanating from the center point at which the arc will be initiated. The user is then prompted for a point on the arc itself. This point defines the radius of the arc. Having entered this point, the user is then prompted for a radial line at which the arc will be terminated. The arc will originate at the starting radial line with a radius equivalent to the point specified on the arc, pass through that point, and terminate at the ending radial line. The direction of the line segments which establish the arc will begin at the starting radial line and pass through the radius point to the ending radial line. If the radial lines coincide, a circle is created.

FILL { OBJECT, SECTION }

The **FILL** command allows the user to create a fillable area out of a set of graphic primitives. The user may fill either an entire editor object or only a section of an object depending on the subcommand he selects for the fill operation. The indicated area will be filled according to the filling attributes specified via the **SET** command.

To increase flexibility the graphics editor allows the user to fill any object (other than an included GL file) he has created, whether or not he had intended to do so when he created the object. Thus, it is not necessary to formally declare an object as a polygon when it is created. Instead, the fill request informally associates all natural terminus points (e.g., line segment terminus points) in the indicated range on the object as verticies of a polygon, and attempts to subsequently fill that polygon as requested. If the verticies do not define a closed polygon due to internal ordering of the data (regardless of how they apear on the screen) the object will be filled improperly or not filled at all. (See **JOIN** command.) Text strings contained within the polygon vertex range are not considered as part of the polygon's verticies and are ignored.

FILL OBJECT *(name)*

The entire fillable area of the object is filled according to the display attribute defaults currently in effect.

FILL SECTION of object *(name)* **from** *(element)* **to** *(element)*

The section between the two selected object elements (lines, characters) is filled according to the display attribute defaults currently in effect.

CHANGING AN EXISTING GRAPHICS OBJECT

After a graphic object is created, its display attributes may be changed by the user via the commands identified in this section. This includes changing the color, linetype, filling properties, and text style of any portion of the object, enlarging or shrinking the object, stretching the object in both the X and Y directions, selectively erasing portions of the object without altering its shape, and changing its location and orientation on the display surface. The process continues until the object's appearance and location satisfy the user.

CHANGE { COLOR, LINETYPE, FILLING, STYLE, HIGHLIGHT }

The **CHANGE** command is used to alter various properties of an editor object after it has been created or joined with other objects. The user may change the **COLOR**, **LINETYPE**, **FILLING**, font**STYLE** or **HIGHLIGHT**ing attributes of part or all of an editor object. The user first selects the particular editor object to be changed, and then identifies that portion of the object over which the change is to take place by selecting starting and ending elements which bound the region of change. The internal element order of the object is examined and the display attributes of the indicated portion of the object are changed to conform to the user's request. The order of the starting and ending points relative to the internal element order of the object determines whether the change includes a central section of the object, or completes the last part of the object and wraps around through the first part of the object.

Having selected a **CHANGE** operation, the user selects the object to be changed. He then indicates the starting and ending points of the region over which the change will take effect. For vectors, the starting point is the vector that has been selected; for text, the starting point is the selected character In both cases the touched graphics element will be changed as indicated. The same condition applies to the ending point. The entire vector or character selected is changed. Additionally, all characters or vectors in between the two are changed. For filled areas, the indicated areas are changed to the desired filling type. (This area must have already been identified via a fill command whether or not visible filling has been used.) After specifying the region of change, the remainder of the command is identical to that described under the **SET MODE** command. For example, **CHANGE color of (object) from (element) to (element) to (saturation) (hue)** changes the color of the elements in the specified range to the requested color values.

ERASE object *(name)* from *(element)* to *(element)*

The **ERASE** command allows the user to erase portions of an editor object while preserving the characteristics and relationship of the remaining elements of the editor object. Thus, the user may eliminate text characters from the middle of a text string or erase selected vectors from a sequence of vectors while maintaining the positional correspondence and display attributes of the other graphic primitives.

Having indicated that an **ERASE** operation is desired, the user selects starting and ending elements of the editor object which bound the region to be erased. These elements and all elements in

between (with respect to the internal element ordering established when the object was created) are no longer visible on the display surface after the operation is completed.

MOVE object *(name)* from *(x,y pair)* to *(x,y pair)*

The **MOVE** command allows the user to reposition a previously created graphics editor object in the graphics editor work area. Having issued a **MOVE** command, the user is prompted for the object to be acted upon. After the user identifies the object, he indicates a reference point associated with the unrepositioned object, the **from** location. The user next indicates a second reference point to be associated with the repositioned object, the **to** location. The object is repositioned within the graphics editor work area such that the spatial relationship of the object to the first reference point is transferred to the second reference point. The object is reclipped as required by its new location in the editor work area.

SCALE object *(name)* by *(factor)* around *(x,y pair)*

The **SCALE** command allows the user to increase or decrease the size of a graphics editor object while preserving its aspect. The user first selects the object to be scaled. He next enters the desired scaling factor at the editor command terminal. Values smaller than 1.0 result in the object being shrunk in size while values larger than 1.0 result in the object being increased in size. The user next selects the point around which the object will be scaled. If this point is exterior to the object, the object will not only change in size, but may shift away from or toward this reference point, depending upon the scaling factor. If the point is interior to the object, then the object will shrink or expand around the reference point. If the point coincides with an edge of the object, then the object will be shrunk toward or expanded away from that edge. Elements of the rescaled object which are no longer contained within the editor work area are clipped as necessary.

SIZE { HEIGHT, WIDTH, BOTH } dimensions of object *(name)*
changing distance from *(x,y pair)* to *(x,y pair)*
to distance from *(x,y pair)* to *(x,y pair)*

The **SIZE** command allows the user to distort an object in both the X and Y directions depending upon the attributes he selects for the subcommands. In utilizing the **SIZE** command the user scales the X or Y components of an object by relating the X and Y components of one line to those of a second line. The object will be sized in the specified direction according to the length-ratio of the first line to the second line in the direction(s) selected. If, for example, the X-component of the first line is larger than that of the second line, and the object is being changed in the horizontal direction, then the width of the object will be reduced by the ratio of the second value to the first. The same applies to the Y-direction. As part of the sizing operation the object will be repositioned on the viewing surface such that the starting point of the first line corresponds to that of the second line.

ROTATE object *(name)* { CLOCKWISE, COUNTERCL }
by *(number)* degrees around *(x,y pair)*

The **ROTATE** command allows the user to rotate an object around an arbitrary reference point, in a clockwise or counterclockwise direction by a specified number of degrees. The object may consist of lines, arcs, or filled areas, but should not contain any textual data. If it does, unpredictable results may occur with regard to the positioning of the textual information.

Having selected a rotation operation, the user first indicates the object to be acted upon. He next indicates the direction in which the object is to be rotated around the rotation-reference point, either **CLOCKWISE** or **COUNTERCLOCKWISE**. The user then indicates on the TTY the number of degrees by which the object is to be rotated. Lastly, he specifies the rotation reference point itself. The object is rotated as specified.

INCORPORATING EXISTING GRAPHIC FILES

Existing graphic files may be incorporated into the file currently being developed by employing the **READFILE** command and the **FILEAREA** option of the **SET** command in conjunction with the scope selection mechanism. As already indicated, the **SET FILEAREA** command selects a file playback window on the viewing surface while the **READFILE** command reads a file into that playback area in the currently selected logical scope.

READFILE " *(filename)* "

The **READFILE** command reads the specified GL graphic file into the currently selected graphics editor logical scope. If the connected version of the backend supports a correspondingly numbered scope, the file is directed to the named GL scope; otherwise, it is directed into the default GL scope. The graphic file is played back into the current file playback window established via the **SET FILEAREA** command. If no playback window was defined in the **SET FILEAREA** command, the entire graphics editor work area is used. Upon file playback, all text and graphics are scaled with respect to the file playback area.

After selecting the **READFILE** command, the user is prompted by a " " " for the filename desired. He enters the desired filename following the quote prompt, terminating the string with another quote character.

RESTRUCTURING EXISTING OBJECTS

Sometime after an object has been created, the user may desire to divide it into two or more smaller objects in order to rearrange the object's structure (e.g., repositioning chunks of text with respect to one another). Alternatively, he may wish to join it with another object to allow subsequent commands to act on both simultaneously. The former can be accomplished via the **BREAK** command while the latter is effected with the **JOIN** command.

BREAK object *(name)* at *(element)*

The **BREAK** command is used to split a graphic object into two parts. The two pieces can subsequently be manipulated independent of one another (i.e., they each become a separate editor object). As indicated earlier, an editor object is a linear list of graphic elements acted upon as a single entity. The object is acted upon in terms of the natural ordering from its head to its tail. Selection of the point on the object at which the separation is to take place identifies those elements of the graphic object which are to be included in one part or the other. Lines and text characters are indivisible entities to the graphic editor. Thus, a line is not split in the middle; rather the line segment touched becomes the first line segment in the new object. Elements specified subsequent to that line segment (during object creation) become associated with the object containing that line segment. Elements specified prior to that line segment are associated with the second object. Similarly, if a text character is touched, it and all elements subsequent to it become associated with one object and all elements preceding it with the other object. Any object display attributes associated with individual object elements remain the same.

JOIN object *(name)* to *(name)*

The user may desire to merge several existing graphics editor objects into a single object to cause subsequent commands to act on the objects collectively. He does this by issuing a **JOIN** command. Having indicated that a **JOIN** operation is desired, the user selects two objects, the first of which is to be appended to the second (follow the second in terms of the internal ordering of the composite object). This ordering is relevent to the actions of other commands such as the **CHANGE** command and should be considered during the joining of objects. The two objects are subsequently treated by the editor as a single object.

DELETING EXISTING OBJECTS

Graphic objects may be deleted individually or collectively using the **DESTROY** or **CLEAR** commands described below. The former operates on individual objects while the latter acts on the collection of objects associated with a logical scope.

DESTROY object *(name)*

The **DESTROY** command allows the user to delete an entire editor object from the editor work area and its internal data base. Having typed a **DESTROY** request, the user identifies the object to be destroyed. The editor then purges that object from its internal data base.

CLEAR { SCOPE, ALL }

The **CLEAR** subcommand allows the user to purge either a specific scope or all logical scopes of any graphics editor information associated with them.

CLEAR SCOPE number: *(integer)*

Erase all information associated with the named logical scope from the display surface. This is particularly useful when dealing with multiple overlays with individual overlays associated with specific scopes. If the user specifies a scope number of "0", the currently selected scope is cleared.

CLEAR ALL

Erase all scopes.

OUTPUTTING THE FILE

There are several commands associated with outputting graphics objects into a graphics file for subsequent playback or use by another program. These commands involve identifying the intended filename, outputting graphic objects into the identified file, and making the file available to the operating system.

OPENFILE " *(filename)* " " *(comment)* "

The **OPENFILE** command is similar to that supported in GL. It allows the user to initiate graphic file output by specifying first a full filename, and second, a comment string which will be placed in that file. The file is opened for subsequent GL output via the **WRITE** command.

WRITE scope number: *(integer)*

The **WRITE** command outputs all editor objects associated with the specified logical scope to the graphics file referenced in the immediately preceding **OPENFILE** command. The user is prompted for the graphics editor logical scope that he wishes to copy into the graphics file. This operation may be performed as often as desired for any of the logical scopes known to the editor in which data is contained. All information so identified will appear in the output file as many times as requested. If the user specifies a scope number of "0", the currently selected scope will be written.

CLOSEFILE

The **CLOSEFILE** command closes the current output file, making it available to the editor or other GL application programs. When the user has completely specified all of the output he wishes to appear in the GL file, he issues a **CLOSEFILE** command to make the file available to the operating system. The file appears in the currently connected directory under the same name the user specified in the **OPENFILE** command.

TERMINATING THE SESSION

When all the files the user wants to create during the current session have been completed, the user terminates the session. He does this by issuing a **QUIT** command, described below.

QUIT

The **QUIT** command requests the editor to close all device connections and terminate execution. All data being used by the editor and all pictures generated by the user are lost except for those explicitly saved in graphics files for subsequent use or recall via the **OPENFILE**, **WRITE**, and **CLOSEFILE** commands. Control is returned to the operating system.

APPENDIX

The following dialogue is indicative of one used to start up the graphics editor.

```
Do you have a device parameter file describing your graphics
terminal? (Y or N):

N

Type the digit corresponding to the graphics terminal to be used:

  1 - TEKTRONIX 4010, 4012, or 4014
  2 - TEKTRONIX 4027
  3 - HEWLETT PACKARD 2647A or 2648A
  4 - HEWLETT PACKARD 9872
  5 - ADVANCED ELECTRONIC DESIGN 512

5

Are the graphics to be generated on some other host?  (Y or N):

N

Type the digit corresponding to how the graphics terminal
is attached to this host:

  1 - to a WILD-TIP port
  2 - to the host directly
  3 - to some other host (host-to-host via NCP)
  4 - to some other host (host-to-host via TCP)
  5 - to a special KI-10 DMA interface

2

Enter the local address (e.g., TTY107:) followed by a carriage-
return:

TTY41:

Do you wish to store this information in a device parameter file?
(Y or N):

N

Specify editor command TTY:

TTY:
```

(You're ready to start working!)

REFERENCE

[Bisbey 80] Bisbey, R., II, D. Hollingworth, and B. Britt, *Graphics Language*, USC/Information Sciences Institute, TM-80-18, 1980.


4-
DT