Network Working Group
Request for Comments: 493
NIC: 15358
References: 282, 258
Obsoletes: 292

J. Michener
MAC
I. Cotton
MITRE
K. Kelley
U. of Ill.
D. Liddle
Ownes Ill.
E. Meyer
MAC

GRAPHICS PROTOCOL

Introduction

This document reflects opinions expressed and decisions reached at the second meeting of the Network Graphics Group, held at the Stanford Artificial Intelligence Laboratory in late November 1971. It describes part of a proposed Network Standard Graphics Protocol for transmitting graphics data within the ARPA network. The particular aspects of the protocol covered in this document relate to the form and content of graphics information sent from a source of graphical information (an application program, say, in the "Serving Host") to a display package for output to a graphics console (at the "Using Host"). This will take the form of a sequence of 8-bit bytes, and will be called the graphics output byte stream.

This document is intended to serve as a basis for discussion and for experimentation over the network. This document does not include form or content of graphics input (data sent from the Using Host to the Serving Host) nor does it cover how the connection is established between the hosts. A proposal for the former will be generated eventually by this committee; the latter is the job of the Connection Committee (of the Network Graphics Group).

This RFC describes the commands which are available in the protocol in terms of the effect they would have at the receiving (Using Host) end. Clearly, some subroutine package is desirable at the Serving Host for use by applications in transmitting graphics data, but on this topic this RFC does not intend to comment.

It may be observed by the reader that no facility is specified in this protocol allowing the Using Host to report logical errors in the graphics output byte stream to the Serving Host. Such a facility would have to be integrated with the graphics input byte stream, since it involves most of the problems related to synchrony of independent hosts.

Background

The reader should probably peruse RFC 282: "Graphics Meeting Report" by Mike Padlipsky to obtain some of the framework surrounding this discussion of network graphics. Also it might be valuable to make note of the model described in RFC 285: "Network Graphics" by Donald Huff.

Levels and Ground Rules Pertaining Thereto

Functions within the graphics protocol will be classified into a number of levels depending partly on how difficult it is to implement those functions. It is intended that any host which claims to implement the functions of level N must implement all lower levels as well. Thus, it is envisioned that sites will implement levels inclemently. Implementations will be improved as a continuing process to include more and more functions, and it is intended that each implementation will be able to identify its own level to a graphics protocol at a remote site which is requesting a graphics interchange. A side result is that each site will be able to determine its own priorities in committing programmers to the graphics protocol as opposed to other efforts.

It is also our intention that implementation of level N will require no knowledge of level N+1. Thus a site can implement a level in the (reasonably) firm knowledge that no changes at higher levels will alter the level implemented. At some time it may be decided by the Network Graphics Group to redefine a level which has previously been firmed up. It is not our intention that this shall happen but one must recognize that the proposed Graphics Protocol is experimental and may have to be changed.

One further ground rule: a stream of commands and data which is valid at a given level, K, shall produce "identical" results on any interpreter of level K or higher. By this we mean that as long as the commands and data take advantage only of strictly defined operations, similar pictures should result. Aspects of the protocol which are not strictly defined (at this time) include character size, character position relative to the beam, how control characters in text output affect the terminal and what happens when the beam is moved or a line drawn outside of the logical screen boundary. This rule forces upwards compatibility, so that an application written using features of a low numbered level will still work at sites which have moved on to implement higher levels. Additionally, any aspects

of this protocol which are explicitly "left unspecified" in the detailed operations descriptions below shall be explicitly specified in any public description of an actual implementation.

We now describe the framework which will be common to all levels.

Basic Data Forms

Information in the Network Standard Graphics Protocol will be expressed as a sequence of 8-bit bytes. A command will consist of a command byte followed by zero or more arguments. The same command byte will always take the same number of arguments in the same form. The length of each argument may be fixed or variable depending on the argument.

A simple type of argument is a "value", which is a 8-bit integer. Another type of argument is a "string" which is a count followed by (count) number of 8-bit bytes. If the count is between 0 and 127, it is sent in a single byte. If the count is between 128 and 2**15-1 (**means exponentiation), it is sent in two bytes with the high order bit of the first byte set to one. The first byte contains the seven high order bits of the number and the second byte contains the eight low order bits. A string is the only type of argument of a command which can vary in length. For example, whenever a command has optional arguments, they will be represented inside of a string.

Coordinate data engendered considerable discussion at the second Network Graphics Group meeting. It was decided that a two-dimensional Logical Coordinate System was required, and each interpreter for the graphic command byte stream would be responsible for mapping this coordinate system to physical device coordinates. It was decided that data in the logical coordinate system would be in twos-complement notation, that it would be fractional, that each edge of the screen would have unit length, and that the origin would correspond to the center of the screen on the output device. The vertical (horizontal) edges of the screen of the output device correspond to the lines X (Y) = -1/2 or X=+1/2-e where e is a small positive number determined by the precision of the fractional data. Particularly the points (-1/2, -1/2) (-1/2, 1/2-e), (1/2-e, -1/2) and (1/2-e, 1/2-e) shall be visible points at the corners of the logical screen. (In the case of a non-square display surface, the implementer may take his own decision. Thus we shall say that the Logical Coordinate System contains points whose coordinates range from -1/2 to a little less than +1/2.

Commands which take coordinate data will be available in various modes. In absolute mode, a position is specified by giving its coordinates in the Logical Coordinate System. In relative mode, the difference between the coordinates of the position and the coordinates of the current position must be specified. Thus a coordinate datum which is an argument for an absolute mode operation should be in the range -1/2 to +1/2-e, while one for a relative mode operation should be in the range -1+e to +1-e.

Interest was expressed at the second Graphics Group Meeting in eventually allowing a very large coordinate space (many bits of precision in each fractional coordinate). This is to be done by permitting the length, in 8-bit bytes, of each coordinate datum to be set (as a mode). It was decided at the meeting that two bytes per coordinate would suffice for now. Thus "e" in the above discussion is 2**(-15) (one in the least significant bit of a 15-bit plus sign fractional coordinate).

Text data will be transmitted as an argument of various commands for display on the output device. Network ASCII will be used to represent characters. At the lowest-levels of the protocol only one character size will be available -- whatever is "normal" on the display device. If the device has no "normal" size, 72 characters per line would be desirable. At higher levels the size of each individual character can be specified.

Also, at the lowest levels, control characters will be passed along to the device for it to do the best it can. However, the consensus of the graphics meeting was that at some reasonably low (but non-zero) level, carriage return, line feed, and backspace should be interpreted to do the right thing.

Picture Subroutines and Related Topics

At the second Network Graphics Group meeting, it was decided that two sorts of picture subroutines were desirable, the primary distinction between them being relative difficulty of implementation. At the meeting, the simpler variety was called a subpicture, and the more complex was called a subroutine. This author believes that these terms do not embody enough semantics to facilitate keeping the two straight and so proposes to standardize on "simple subroutine" and "full subpicture" instead.

The only parameter which can be passed to a simple subpicture is the "current beam position". In other words, if such a subpicture is called more than once in a picture, the only difference in appearance

between the various calls is a translation due to the beam position at the time of the call. Full subpictures, on the other hand, take parameters which can cause scaling, rotation, reflection, or anything else we come up with.

It is planned that a subpicture definition need be transmitted only once (per network connection) and would not be deleted by a "new picture" operation. Thus a changing picture could be subdivided into several parts on a basis of static versus changing information; only definitions of parts which change need be transmitted to redraw the picture.

Traditionally, picture subroutines which depend only on the initial beam position have been restricted to relative data mode drawing operations. In view of the fact that subpictures will probably be used to save static picture information, it is desirable to allow absolute data mode operations in simple subpictures.

The next question naturally arises -- what does absolute data mean in a full subpicture which takes both position and scale parameters? Is absolute data really absolute in this case? This author believes that the answer is as follows: the full subpicture is really a picture in its own right, so it has its own logical coordinate system, and its absolute data is really within this coordinate system. Thus "shifting and scaling" a full subpicture really means "scale the subpicture in its own coordinate system and shift the result as a whole".

In summary, then, a major difference between simple and full subpictures is that a full subpicture has its own logical coordinate system and a simple subpicture uses the logical coordinate system of whoever calls it. This distinction is the reason why full subpictures are harder to implement than simple subpictures.

Another point discussed at the meeting was a special data mode whereby a subpicture can display data at absolute positions on the screen, i.e., absolutely in the main (picture) program. To achieve this, the meeting proposed special data modes for the three operations: move beam invisibly, draw line, and display dot. The intent of these data modes was to bypass all rotation, scaling, and clipping functions associated with the current level of subpicture nesting until this mode was cleared in a certain way. This same effect can be achieved more directly and implemented more efficiently by two commands: one to save and one to re-establish the logical coordinate system for the current subpicture. (Additionally, of course, the "save" operation would establish the initial, highest level, logical coordinate system.)

Simple Subpicture Calls

Besides the <identifier> of the subpicture to be called, a simple subpicture call may specify two optional parameters; the first is an <identifier> which is the "name" (in a sense described below) of this particular subpicture call and the second is an absolute position on the calling page to be invisibly moved to, prior to calling the subpicture. When (eventually) the viewer is allowed to interact by "picking" information displayed before him, if the information is part of a subpicture, then the "name" of the subpicture call will be part of the "graphic input" reported to the serving host. If the information picked by the viewer is within several levels of subpicture calls, the names of each of the calls will be reported in a manner which indicates their nesting. (Note that just the name of the subpicture by itself is not sufficient, since one subpicture may be displayed in several positions and the application may wish to distinguish between individual calls.) If the identifier is not specified it defaults to the null string. If the position (for the invisible move) is not specified, the current beam position is used.

Which of these two parameters are present is encoded by two bits in a code byte which precedes the parameters. If both parameters are present then they are always in the same order; this order and the bits of the code byte assigned to the two parameters are specified in the detailed description of the Simple Instance command (and in the BNF in Appendix 1). Preceding even the code byte, and immediately following the name of the subpicture which is being called upon, is a count of the data in the remainder of the instance command. Thus is included so that it is not necessary to decode the code byte to determine the total length of any one Simple Instance operation.

Windowing: Clipping, Blanking, or (?)

While on the subject of coordinate systems and subpictures, it might be good to touch on the topic of: who (which end of the connection) is responsible for doing what, when a picture is potentially going to be displayed beyond the edge of the virtual screen? It was the consensus at the graphics meeting that the interpreter of the graphics protocol (i.e., the using end) would not be held responsible for doing anything reasonable in case a picture displays information beyond the edge of the screen (e.g., by relative moves and draws).

The interpreter must react properly to the next absolute data in the proper range, however. Various solutions to this situation in existing graphics systems include:

clipping a line to display as much as is proper,

blanking the whole of a line if any part is invisible, or

discarding high order bits of the current position register, so that no invisible positions can be represented ("wraparound").

In addition to problems of the edge effects at the highest level, problems arise with respect to (full) subpictures. It is nice to be able to select a rectangular portion of a subpicture to be displayed as part of a subpicture call. (See: Newman, Display procedures, Communications of the ACM, Volume 14, Number 10, October 1971, pp651-660). In accordance with the consensus of the meeting, which was to make this capability optional, this author merely hopes to include in the protocol a method of encoding this information since his site a) can handle some such windowing, and b) hopes to provide a service facility to perform this function.

Appendix 2 describes how to concatenate several levels of portions into a single rectangular test, as long as no rotations are involved. It also outlines the problems related to rotations and portioning.

Full Subpicture Calls

We are now in position to consider what may be specified as part of a full subpicture call, in addition to the name of the subpicture being called, which is, of course, required. The data described below will all be optional: a single code byte will precede all these data; the presence or absence of one of the parameters will be indicated by a bit in the code byte being one or zero. The parameters will always appear in the same order, if they are present. This order is given below in the detailed description of the Full Instance command (and in the BNF in Appendix 1). Additionally, preceding even the code byte, will be a <count> of the following bytes, including the code byte to determine the total length of any particular Full Instance operation.

One parameter is an <identifier> which can be used to distinguish this particular call to this subpicture from all other calls to the subpicture. This parameter was already described under Simple Subpicture Calls.

On parameter which may be specified is a translation: this will be specified by giving the absolute coordinates of the center (on the calling page) of the image of the subpicture; this will default to the beam position current at the time of the call.

A rotation may be specified by giving a 16-bit fraction in the range 0 to .1111111111111111 (binary) inclusive; this fraction will represent what part of a full circle (2pi) the rotation is. The default value of angle of rotation will be zero.

(Actually, the rotation representation scheme works identically if one thinks of it as a two's complement fraction from -1/2 to just less than +1/2. That is, the same bit configurations encode the same rotation, due to the periodic nature of sine and cosine. For example, binary zero always represents 0 pi 010000...0 denotes pi/2 in both schemes; 100...00 denotes 1/2 in one scheme and -1/2 in the other, which correspond to rotations of +pi and -pi respectively, i.e. identical rotations.)

Also specifiable as apart of a full subpicture call is a rectangular portion of the called picture to be imaged on the calling picture (see previous section for a discussion of clipping). This rectangle is specified by its center and one half its total size in x and y. That is, the rectangle will consist of all points whose x coordinate differs from that of the center by no more than the specified x-size and whose y coordinate satisfies a similar condition. The default for these values will place the center at the origin and give both the x half width and the y half width the value of +1/2. Thus the default includes the whole of the logical coordinate system of the called page (and also some points one of whose coordinates are +1/2, which, strictly speaking, lie "outside" of the coordinate system; how this inconsistency is resolved is left unspecified).

Finally, one must specify the scaling to be applied in determining the image; this can be done in many ways. One way is to specify a uniform magnification to be applied to the subpicture. So that magnifications in a wide range can be achieved, it is the author's opinion that some form of scientific notation (i.e., floating point) will have to be employed. If there is already a network standard floating point notation (which I am not aware of) it should be employed. Failing that, it is suggested that this notation consist of an 8-bit (two's complement) exponent followed by a 16-bit (two's complement) fractional part.

Another form of scaling is to specify separate magnifications in x and in y, to be applied to the subpicture before any rotation is performed. Yet a third way is to specify a rectangular area in the calling picture's coordinate system to be filled with the image of

the subpicture. Since the center of the image is already specified (by the translation), this image information consists only of half-edge size data. If none of the three methods of scaling are chosen (and an affine transformation (see below) is not given explicitly), then a uniform magnification of unity (i.e., no scaling) is used. Note that the three forms of scaling tend to contradict each other and only one of them should be used in any one call. What happens if self-contradictory information is given in these fields is left unspecified.

Appendix 2 presents the mathematics involved in transforming the subpicture's coordinate system into the calling picture's coordinate system. It is shown there that all the individual operations (scaling, rotating, and translating) can be represented as a single affine transformation (which consists of 6 values). It might be nice to permit the serving program to specify this transformation directly. Accordingly, one possible parameter of a full subpicture call will consist of six floating point numbers (of the form described under magnification, above) to be interpreted as an affine transformation. Indeed, if the affine transformation has the following form:

```
/_ |x  |y_/ = /_ x y_/ * / L11 L12  / + /_ T1 T2_/
                         /_ L21 L22 _/
```

then the values shall (arbitrarily) be sent in the following (columnar) order: L11, L21, L12, L22, T1, T2. This affine transformation should be invertible that is, L11*L22 - L21*L12 should not be zero.

Viewporting

Another topic discussed at the meeting, and referred to the protocol committee for decision, was the capability of placing the "top level" picture in some rectangle of the virtual screen. The default rectangle might be the full screen. Alternatively it might be left up to the viewer to specify the default (via) interaction with the graphics system at the Using Host). In general, viewporting allows more than one "top level" picture to be viewed at once. The desire to view several different pictures on the same screen arises in cases where multiple users are working together and in cases where one user is interacting with a group of applications (in separate serving hosts). This author maintains that the coordinate transformations required by this feature are simpler than that of "full subpictures" since no rotations are involved, and would be part of the same mechanism in its implementation. In particular, merely another affine transformation (see Appendix 2) would be added to the levels caused by full subpicture calls. All that is required is keeping

track of viewport identifiers and the associated rectangles. Since little extra work is involved, it is proposed that this feature be included at some high level of the protocol.

Command Codes

Each command in the graphics protocol will be assigned a non-negative value which will represent this command in the byte stream. The algorithm whereby values and commands are associated is, it turns out, a very touchy subject. There are five or ten different criteria for a "best" algorithm, each criterion different in emphasis. This Gordian knot will be cut, in this proposal, by ordering the commands approximately according to level, and then just numbering them. In addition, if several closely related commands occur at the same level, some attempt will be made to encode variations of meanings in terms of bit configurations. Even if some later consideration causes a change in ordering to be proposed, it is this committee's feeling that the numbering should not be altered. However, until this matter is firmly settled, it is strongly advised that any implementation take into account the possibility of reassignment of command codes.

Particular Proposal for Level 0 Protocol

It is proposed that level 0 be kept very simple. This is so that implementation can be quickly accomplished and experimentation with the protocol begun. Another reason is that the least powerful host and even programmable terminals should be able to implement it. In accordance with this, the "rule" was made that a command be included only if its output is a function solely of the current command and the "beam position" current at the start of the command. In other words, the interpreter for level 0 need have no internal storage for "modes" or pushdown stacks. With this restriction it is hoped that a very simple implementation will be possible for level 0. In particular, perhaps one could eventually build a hardware translator from level 0 code to one's own particular terminal's code.

Note that in the opcode assignment for level 0, bits 4, 2, and 1 have special meaning for the move, line, and dot commands. In particular, the 1 bit encodes absolute versus relative data mode, the 4 bit encodes whether any visible output occurs, and the 2 bit determines whether the visible output is a line or a dot.

Level 0: Command Summary

The following is a list of commands (and their syntax) in level zero. Detailed descriptions of these commands follow in the next section. Commands dealing with protocol may be added by the Connection Committee. (They currently request opcodes in the range 128 to 255.)

(As described in Basic Data Forms, above, <x coordinate>, <y coordinate>, <x delta> and <y delta> are two-byte coordinate values, <string> is a count followed by <count> many bytes and <value> is an eight bit number.)

Decimal	Octal	Binary	Format
0	0	00000000	Null
1	1	00000001	Erase screen and reset beam
2	2	00000010	Move Absolute <x coordinate> <y coordinate>
3	3	00000011	Move Relative <x delta> <y delta>
4	4	00000100	Draw Absolute <x coordinate> <y coordinate>
5	5	00000101	Draw Relative <x delta> <y delta>
6	6	00000110	Dot Absolute <x coordinate> <y coordinate>
7	7	00000111	Dot relative <x delta> <y delta>
8	10	00001000	Text <string>
9	11	00001001	TextR <string>
10	12	00001010	End of Picture
11	13	00001011	Escape <value> <string>

Level 0: Command Descriptions

0 Null Statement ("NULL").
This statement has no arguments--and no effect, either.

1 Erase screen and reset beam to origin ("ERASE").
This command indicates that a new picture is about to be drawn. It should always be (eventually) paired with a following End of Picture command.

2 Move beam invisibly to absolute position ("MOVEA") <x coordinate> <y coordinate>.
Nothing is drawn; the beam is positioned to the specified absolute x,y position.

3 Move beam invisibly by relative amount ("MOVER") <x coordinate> <y coordinate>.
Nothing is drawn; the beam is shifted by the specified amount in x and y.

4 Draw line to absolute position ("DRAWA") <x coordinate> <y coordinate>.
A line is drawn from the current beam position to the specified absolute x,y position.

5 Draw line to relative position
("DRAWR") <x delta> <y delta>.
A line is drawn from the current beam position to the position delta x and delta y away.

6 Display a Dot at absolute position
("DOTA") <x coordinate> <y coordinate>.
The beam is moved invisibly to absolute position x,y and a dot is displayed there.

7 Display a Dot at relative position
("DOTR") <x delta> <y delta>.
The beam is moved invisibly by the specified amount in x and y and a dot is displayed there.

8 Display text ("TEXT") <string>.
At the current beam position, display some characters at the normal size for the device being operated. <string> consists of a <count> followed by count many characters. If there is no "normal size", choose the size so that seventy-two characters are displayed per line. The characters in the string are coded in network ASCII all codes between 0 and 127 (decimal) inclusive are permitted. (At level zero, what happens to control characters is left unspecified.) Where the beam is, following execution of this command, is left unspecified, except that another Display Text command immediately following will append its text to the previous string. (The use of the TEXT command is discouraged; use TextR instead.) The position of the first character relative to the initial beam position is left unspecified.

9 Display text and restore beam ("TEXTR") <string>.
At the current beam position, display a string of characters at the normal size for the device being operated; then reposition the beam to where it was before the command. <string> consists of a <count> followed by count many characters. If there is no "normal size", choose the size so that seventy-two characters are displayed per line. The characters in the string are coded in network ASCII; all codes between 0 and 127 (decimal) inclusive are permitted. (At level zero, what happens to control characters is left unspecified.) The position of the first character relative to the initial beam position is left unspecified.

10 End of Picture ("ENDPIC").
This command denotes the end of a new picture. It must be paired with a preceding ERASE command.

11 Escape to device specifics ("ESCDEV") <value> <string>.
If "value" is the code assigned (by the Protocol Committee) to the device being operated, then transmit the eight-bit bytes in <string> (which starts with a <count> indicating the number of bytes) to the

device without examining them. Otherwise ignore this command. If the device does not accept 8-bit information, reformat the data in some device specific way; an example would be throwing away the high order bit for a seven bit device, or gathering 5 8-bit bytes into one 36-bit word, again discarding the high order bits, perhaps. The action of the bytes in the string should leave alone (or at least restore) any hardware beam position registers in the device which the interpreter might conceivably depend on.

This command really should not be used it was included at level 0 so that specific applications can do mode setting and other device specific manipulations. For example, ARDS terminals may optionally have several independently addressable output scopes. The selection mechanism changes state only when a particular sequence of ASCII characters reaches the terminal. Thus ESCDEV would be used to select which scope(s) is/are to be affected by following commands. (The current state is invisible to the graphics package at the Using Host.)

Further, suppose that another make of terminal has a similar option, which responds to a different code sequence. This possibility is the motivation for conditionally ignoring the ESCDEV command based on the "<value>" specified. Given that a particular application will only be used to output to either an ARDS or this second make (with the multiple scope option), then the application could always send two ESCDEV commands, one applicable only to ARDS terminals, and the other applicable only to the second make.

LEVEL 1

*Set Line mode ("LINMOD") <value>.
This command sets the current line mode possible modes and the <value> which sets each are: solid (0), dashed (1), dotted (2), and others (3 or >). At the beginning of a new picture (i.e., after an Erase and Reset command), line mode is solid. If a site does not have a certain mode readily available, it may a) simulate it in software, b) substitute another in its place (dashed for dotted, or vice versa) c) ignore it entirely. What is provided should be clearly indicated in any public document. It is strongly recommend that at least solid and one other mode be provided.

*Set intensity ("SETINT") <value>.
This command sets the intensity of lines, dots and characters displayed following the command. If <value> is 128 decimal, normal intensity should be set. If <value> is 255 decimal, brightest should be selected, and if it is 0, then the beam should be blanked. Intermediate values should be mapped appropriately as the implementer

sees fit. For instance, if brightest is the same as normal, all values from 128 through 255 should be mapped to normal. Information displayed between the start of a new picture (the ERASE command) and the first SETINT command appears at normal intensity.

*Text out ("TEXTO") <string>.
Starting from the current beam position, this command displays the <string> (of network ASCII characters) formatted as if it were typed material (at the current intensity). <string> consists of a <count> followed by count many characters. That is, text extending past the right margin will be broken and repositioned at the left margin on the next line down. Of the control characters, only carriage return, line feed, and backspace are required to be interpreted properly.

*Subpicture header ("SUBHED") <identifier> <count> <header info>.
This command begins the definition of a subpicture named "<identifier>". This definition is terminated by a matching SUBEND command. The definition will be remembered until a new one is specified or until the graphics network connection is broken. Note that <identifier> is a <string> consisting solely of capital letters and numbers.

Subpicture definitions may be nested this will be equivalent to transmitting the two definitions separately. In other words, all subpicture names are globals and are "known" to all other subpictures. If a subpicture definition has not been received prior to its use in a picture, the empty subpicture should be displayed in its place until a definition is received.

A subpicture definition need not be transmitted as part of a picture (i.e., within an ERASE and END command pair). Indeed, all subpicture definitions might precede the main picture.

Currently, the <count> will always be 1, indicating only one byte of <header info> follows, but at higher levels of the protocol room for expansion may be required. In the <header info>, the 80 hex bit will be set if this subpicture can be a simple subpicture, and the 40 hex bit will be set if the subpicture can be a full subpicture. (It is possible that one subpicture can be both.)

Other information that may eventually be present in <header info> include whether the current value of a certain mode or parameter should be saved on entry to, and restored on exit from, this subroutine whenever it is called. These modes and parameters include: line mode, intensity, character size, and data length.

*Subpicture end ("SUBEND").
This command ends the definition of a subpicture. Each SUBEND must match a preceding SUBHED command.

*Simple instance ("INSTS") <identifier> <simple instance tail>
This command indicates that the subpicture <identifier> is to be called (instanced). At this level, level 1, no subpicture may call another; if one does, what happens is left unspecified. Also, this must be a call to a simple subpicture. Thus the 80 hex bit of the single byte of <header info> must have been set in the SUBHED command which started the definition of <identifier>. If the subpicture <identifier> has never been defined, the empty subpicture should be displayed in its place.

The <simple instance tail> begins with a count of the amount of information which follows. This count may be zero. If non-zero, the next byte is a code byte to be interpreted to see what further information follows. If the 80-hex bit is set, next in the byte stream is an <identifier> (called "AS information"). This <identifier> is the name of this particular instance of the subpicture as described under Simple Subpicture Calls. If the 40-hex bit is set, then next in the byte stream (following the AS information, if present) is an x,y position (in the calling picture's coordinate scheme) at which the subpicture will be centered. (This is called AT information.)

If AT information is not specified, the current beam position is used as a default. If AS information is not specified, it defaults to the <string> containing zero characters. If neither the 40 hex nor the 80 hex bits are set, then neither the AT information not the AS information is present, and the code byte should be zero. (Also, the length count had better be 1.)

Changes to levels 0 commands for level 1.

TEXT and TEXTR -- Carriage return, line feed and backspace characters should definitely be interpreted whenever they appear in <string>. The results of other control characters remain unspecified. The intensity of the characters shall be affected by the SETINT command.

ERASE -- Normal intensity and solid line mode must be established at the start of a new picture.

DRAWA and DRAWR -- Line mode and intensity shall be affected by the LINMOD and SETINT commands.

DOTA and DOTR -- Intensity shall be affected by the SETINT command.

LEVEL 2

*Mark ("MARK").
This command causes the current x,y beam position to be saved on a pushdown stack. This pushdown stack must be separate from the subpicture call pushdown stack.

*Move to mark and pop ("MOVEMK").
This command sets the current beam position equal to the x,y position at the top of the "mark" pushdown stack. If the stack is empty, the origin is used, instead. Then the stack is popped up (unless it is empty).

*Draw to mark and pop ("DRAWMK").
If the "mark" pushdown stack is not empty, this command draws a line (of the current line mode and intensity) from the current beam position to the x,y position at the top of the "mark" pushdown stack, and sets the beam position to that value. Then the stack is popped. If the stack is empty, the line is drawn to the origin and the beam position is set there also.

Changes to level 0 and 1 for level 2.

INTS -- arbitrary levels of simple subpictures must be supported. (Note that recursive use of subpictures is not allowed: once recursion starts, it can never be stopped.) The pushdown stack for subpicture calls must be kept separate from the "mark" pushdown stack.

Level 3

(Perhaps all rotational transformations should be put at a higher level, for instance higher than viewport operations.)

*Full Instance ("INSTF") <identifier> <full instance tail>
This command indicates that the subpicture <identifier> is to be called (instanced) in a "full" manner as described in an explanatory section. For one thing, this means that the 40 hex bit of the single byte of <header info> must have been set in the SUBHED command which started the definition of <identifier>. If <identifier> has never been defined, the empty subpicture (i.e., nothing) should be displayed in its place.

The <full instance tail> is similar to the <simple instance tail> described under the INSTS command, but the former contains more information. Below is a list of the information which can be specified, and the bit assigned to the presence/absence of each piece of information. The pieces of information which are present always

appear in the byte stream in the order they are described in this list. (All pieces of information are described more fully in Full Subpicture Calls, except for the "AS information" which is described in Simple Subpicture Calls.)

Bit (hex)	Information
80	As information --"name" of this particular instance. Consists of an <identifier>.
40	Translation information -- Center of the subpicture's image on the calling page. Consists of an <x coordinate> and a <y coordinate>.
20	Rotation -- Fractional part of 2pi to rotate the image counterclockwise. Consists of a 16-bit unsigned fraction.
10	Portion Information -- Rectangular part of subpicture which is to be displayed. Consists of <x coordinate>, <y coordinate>, <x delta>, and <y delta>.
8	Uniform Magnification -- Amount to scale the whole subpicture. Consists of a floating point number (which should not be zero).
4	Separate x and y magnification -- Separate scales for the x and y axes of the subpicture. Consists of two floating point numbers (neither of which should be zero).
2	Image Size -- How large a rectangle on the calling page is the image to occupy. Consists of an <x delta> and a <y delta> (neither of which should be zero).
1	Affine transformation -- The map from the called to the calling coordinates system. Consists of six floating point numbers.

Notes:

1) At most one of the three bits: 8, 4, and 2, should be set.

2) If the 1 bit is set, bits 2, 4, 8, 20, and 40, should not be set.

3) If additional optional parameters are ever added to the full subpicture call, another code byte could follow all the above information. In that case, the <count> part of the <full instance tail> would include this second code byte and any additional bytes of information.

*Escape to top level coordinate system ("ESCTOP").
Until a RESLEV command is (subsequently) executed, all display commands (moves, draws, dots, and texts) shall operate as if they were issued by the top level (main) picture instead of the subpicture containing them. That is, they shall be mapped to the screen according to the map for the highest level. Subpicture calls themselves, which are made while an ESCTOP command is in effect, are not affected by the command. That is, transformations are calculated as if the command were not in effect. The calculated transformations are ignored, however, and information displayed by the subpicture still appears to be at the top level, until a RESLEV command nullifies the ESCTOP mode. Thus a subpicture call executed while an ESCTOP command is in effect, acts as if a RESLEV were executed immediately before the call, and an ESCTOP command were executed as the first command of the subpicture. Similar considerations hold for returning from subpictures.

*Resume current level coordinate system ("RESLEV").
This command restores the logical coordinate system corresponding to the subpicture currently executing, in case that coordinate system was disabled by an ESCTOP command. (See ESCTOP.)

Changes to levels 0, 1, and 2 for level 3.

MARK -- the saved beam position shall be in terms of the logical coordinate system, not the physical coordinate system.

TEXTR, TEXT, TEXTO -- Since a full subpicture is supposed to be transformed as a whole, as if it were a picture in its own right, it appears to this author that, in particular, all beam movements related to characters should be affected. This includes character size, tab, carriage return and line feed. In particular, carriage return should set the beam to the left margin--that is, to the left edge of the logical coordinate system of the called subpicture. All these changes may be very hard to accomplish, and what should be done will be left unspecified at this time, with comment from readers particularly invited.

Level 4

(Perhaps viewpoint operations can be included in level 3.)

*Declare Viewport
("SETVW") <viewport id> <x coordinate> <y coordinate> <x delta> <y delta>
Set the viewport identified by <viewport id> to represent the indicated area of the logical screen. The x and y data are not physical screen coordinates, since that would involve device

dependencies. This command completely supersedes any previous declaration of the same viewport. If information is already displayed within the viewport specified, this command causes the displayed information to be relocated on the screen to its new position.

If the area specified exceeds the limits of the graphics standard display screen, what happens is left unspecified. Viewports need not be disjoint; in other words, two viewports can present display information at the same point on the screen.

If <x delta> or <y delta> are negative, the viewport named should be deleted. All information displayed by it shall no longer appear.

Because it affects the top level picture, this author feels that this command should not occur as part of a picture or in a subpicture declaration.

*Add subpicture to viewport ("ADDSVW") <identifier> <viewport id>
The subpicture named <identifier> is displayed within the viewport specified, if it is not already displayed there. (If it is, nothing is done.) The subpicture must be capable of being called via a full subpicture call. If the viewport has never been declared via a SETVW command what happens is left unspecified. (Three possibilities are: nothing is displayed; the viewport defaults to the whole logical screen; the human viewer is permitted by the Using Host to specify the viewport.) If the viewport is subsequently declared, the subpicture shall be displayed in it. If the subpicture has never be declared, nothing is displayed for it; when and if it is subsequently declared, the new definition is displayed in the viewport. More than one subpicture may be displayed in a single viewport at once.

Because it affects the top level picture, this author feels that this command should not occur as part of a picture or in a subpicture declaration.

*Clear viewport ("CLVW") <viewport id>
All subpictures which have been added with the ADDSVW command to the viewport specified in this command are removed from it. Thus the specified viewport contributes nothing to what the human viewer sees. (After a CLVW, the area of the viewport may not be blank due to other, non-cleared viewports which overlap it.)

Because it affects the top level picture, this author feels that this command should not occur as part of a picture or in a subpicture declaration.

Changes to levels 0, 1, 2, and 3 for level 4.

ERASE -- All viewports are cleared (as in the CLVW command) but their declarations are remembered.

ENDPIC -- This command partially loses its purpose: it no longer serves to mark the end of all picture information to be presented to the user, since viewport operations may follow which amend or alter the picture. This function is partially taken over by the DELAY and NODELAY commands described below.

Level ?

*Set Character Size ("SETCHS") <x delta> <y delta>.
Until further notice, characters shall be displayed so that each occupies approximately <x delta> and <y delta> in the appropriate coordinate direction in the current logical coordinate system. Inter-character and inter-line spacing could be certain percentages (any ideas?) more than <x delta> and <y delta>, or they could be specified separately. In any case, only a "best effort" would be expected at a site. Character size is always set to normal (as defined by level 0 character size being normal) by the ERASE command. <x delta> and <y delta> should be positive, except that if <x delta> is equal to zero, then <y delta> being negative, zero, or positive, correspond to a character size which is "smaller than normal", "normal", or "larger than normal". How much smaller or larger than normal is left up to the site.

Changes to levels 0 and 1 for level ?.

TEXTR, TEXT, and TEXTO -- Characters are to be displayed according to the current character size.

ERASE -- Must establish normal character size, normal being that for level 0.

Level ?′

*Set Data Length ("SETDLN") <value>.
Until this mode is explicitly changed with another SETDLN, various data will consist of <value> number of bytes. <value> may be 1, 2, 3, or 4. Affected are the following syntactic types (refer to Appendix 1): <coordinate>, <x coordinate>, <y coordinate>, <double coordinate>, <x delta>, <y delta>, <angle>, and the fractional part of a floating point number. When a network connection is initially established, the data length is two.

Level ?''

(These commands should probably be at the same level as viewport operations, if not earlier.)

*Extensive Changes Follow ("DELAY").
This optional command is designed to eliminate futile effort on the part of the Using Host programs. At some hosts and/or with some output devices (particularly storage tubes) a non-negligible amount of time may be required to present an image to the human viewer. If extensive changes are going to be made, this command would be used to prevent the Using Host graphics package from updating the image after every change. A NODELAY command exits from the DELAY mode and causes the image to be prepared and presented to the viewer.

For example, the current picture may display four subpictures each of which is going to be redefined. Without a DELAY command, the viewer would see successive stages of the change, each possibly involving a large amount of computation or transmission time.

*End of Extensive Changes ("NODELAY")
This optional command undoes the effect of the DELAY command.

Appendix 1: BNF for the Graphics Protocol Byte Stream

Key to below:
Non-terminals are represented in < >.
Terminals which are keywords standing for particular eight-bit values are in capitals.
Terminals whose meaning should be clear to the reader are in lower case.
Note that "empty_string" means "zero bytes", not "a <string> whose <count> is zero."

```
<graphics output byte stream> ::= empty_string
            | <picture> <graphics output byte stream>
            | <subpicture declaration> <graphics output byte stream>
            | <viewport operation> <graphics output byte stream>
            | <transmission control stt> <graphics output byte stream>
<picture> ::= <new picture sst> <program stt group> <end picture stt>
<subpicture declaration> ::= <subpicture header stt> <program stt
                                group><subpicture end stt>
<viewport operation> ::= <declare viewport stt>
            | <add subpicture to viewport stt>
            | <clear viewport stt>
<transmission control stt> ::= <set data length stt>
            | <extensive changes follow stt>
            | <end of extensive changes stt>
<program stt group> ::= empty_string | <program stt <program stt group>
<program stt> ::= <picture control stt> | <display stt> |
                 <transmission control stt>
<picture control stt> ::= <escape to device stt>
            | <escape to highest coordinate system stt>
            | <restore coordinate system stt>
            | <mark stt>
            | <null stt>
            | <line mode stt>
            | <set intensity stt>
            | <subpicture declaration>
            | <simple instance stt>
            | <full instance stt>
            | <set character size stt>
<display stt> ::= <move absolute stt>
            | <move relative stt>
            | <draw absolute stt>
            | <draw relative stt>
            | <dot absolute stt>
            | <dot relative stt>
            | <move to mark and pop stt>
            | <draw to mark and pop stt>
```

```
           | <text and restore beam stt>
           | <text stt>
           | <text out stt>
<new picture stt> ::= ERASE
<end picture stt> ::= ENDPIC
<subpicture header stt> ::= SUBHED <identifier> count> <header info>
<header info> ::= 80-hex | 40-hex | C0-hex
<subpicture end stt> ::= SUBEND
<set viewport stt> ::= SETVW <viewport id> <x coordinate>
                       <y coordinate> <x delta> <y delta>
<add subpicture to viewport stt> ::= AADSVW <identifier> <viewport id>
<clear viewport stt> ::= CLVW <viewport id>
<extensive changes follow stt> ::= DELAY
<end of extensive changes stt> ::= NODELAY
<escape to device stt> ::= ESCDEV <device code> <string>
<escape to highest coordinate system stt> ::= ESCTOP
<restore coordinate system stt> ::= RESLEV
<null stt> ::= NULL
<mark stt> ::= MARK
<line mode stt> ::= LINMOD <value>
<set character size stt> ::= SETCHS <x delta> <y delta>
<set data length stt> ::= SETDLN <value>
<move absolute stt> ::= MOVEA <x coordinate> <y coordinate>
<move relative stt> ::= MOVER <x delta> <y delta>
<draw absolute stt> ::= DRAWA <x coordinate> <y coordinate>
<draw relative stt> ::= DRAWR <x delta> <y delta>
<dot absolute stt> ::= DOTA <x coordinate> <y coordinate>
<dot relative stt> ::= DOTR <x delta> <y delta>
<move to mark and pop stt> ::= MOVEMK
<draw to mark and pop stt> ::= DRAWMK
<text and restore beam stt> ::= TEXTR <string>
<text stt> ::= TEXT <string>
<text out stt> ::= TEXTO <string>

<simple instance stt> ::= INST <identifier> <simple instance tail>
<full instance stt> ::= INSTF <identifier> <full instance tail>
<simple instance tail> ::= eight_bits_of_binary_0
                     | <count> <tail code> <as clause> <at clause>
<tail code> ::= bit_pattern_indicating_what_clauses_follow
<full instance tail> ::= eight_bits_of_binary_0
                     | <count> <tail code> <as clause> <at clause>
                     <rotation clause> <portion clause>
                     <uniform magnification clause>
                     <separate magnification clause> <image size
                     clause> <complete transformation clause>
<as clause> ::= empty_string | <identifier>
<at clause> ::= empty_string | <x coordinate> <y coordinate>
<rotation clause> ::= empty_string | <angle>
```

```
<portion clause> ::= empty_string | <x coordinate> <y coordinate>
                     <x delta> <y delta>
<uniform magnification clause> ::= empty_string |floating point number>
<separate magnification clause> ::= empty_string |
                    <floating point number> <floating point number>
<image size clause> ::= empty_string | <x delta> <y delta>
<complete transformation clause> ::= empty_string | six_<floating point
                    number>'s

<angle> ::= 16-bit_non-negative_fractional_part_of_a_circle
<x coordinate> ::= <coordinate>
<y coordinate> ::= <coordinate>
<x delta> ::= <double coordinate>
<y delta> ::= <double coordinate>
<coordinate> ::= signed,_two_s-complement,_fraction_in_range
                    -1/2_to_less_than_+1/2
<double coordinate> ::= signed,_two_s_complement,_fraction,
                    range _strictly_between_-1_and_+1
<floating point number> ::= network_standard_floating_point
                    number_if_any
                    | 8-bit_two_s_complement_exponent_part and a
                    16-bit_two_s_complement_fraction_part <count>
                    ::= 7-bit_non-negative_integer
                    | 15-bit_non-negative_integer_represented_in
                    "excess_2**15"_notation
<string> ::= <count> count_8-bit_bytes
<identifier> ::= <count> count_upper_case_letters_or_numbers
<viewport id> ::= <identifier>
<device code> ::= 8-bit_integer
<value> ::= 8-bit_integer
```

Appendix 2. Mathematical Formulae for Subpictures

Transformations

In this appendix positions in a logical coordinate system will be represented by a row vector with two elements, as in /_ x y_/. Vectors and matrices will be delimited by these funny brackets: /_ _/. Various symbols will be used to represent parameters in a full subpicture call relating to a transformation from one coordinate system to another; these are defined below:

Mx and My : magnifications in x and y to be applied before any rotation.
They may be negative indicating reflection.
A: an angle of rotation in the range 0 to just less than 2pi.
/_ |cx |cy_/ : the center (in the calling picture) of the image of the subpicture.

```
|sx |sy : the half-sizes, in the x and y directions, of the
          image on the calling page in terms of the calling page's
          coordinate system.
          They may be negative to indicate reflection.
/_ x y_/ : a position on the called page.
/_ |x |y_/ : the position on the calling page corresponding to /_x y_/.
/_ Pcx Pcy_/ : The center of the portion of the called subpicture's
               coordinate system which is to be mapped to the calling
               page.
               This defaults to /_ 0 0_/ if not specified.
Psx and Psy : The half-sizes in x and y of the portion of the
              subpicture to be mapped. These both default to +1/2
              in not specified.
```

(If a uniform magnification is specified, set Mx and My equal to it and proceed below as if they were specified.)

If magnifications are specified, the following holds:

```
/_ |x |y_/ = (/_ x y_/ - /_ Pcx Pcy_>) * / Mx/Psx  0    / *
                                          /_   0  My/Psy_/

         / cos 0 sin 0 / * / 1/2 0  / + /_ |cx |cy_/
        /_ -sin0 cos0_/  /_ 0  1/2_/
```

or in other words,

```
1)
/_|x |y_/ = /_ x-Pcx y-Pcy_/ * / Mx cosA/2Psx Mx sinA/2Psx /
                               /_ -My sinA/2Psy My cosA/2Psy_/
            +/_ |cx |cy_/
```

(The factor of 1/2 is necessary because, for instance, (x-Pcx)/Psx ranges from -1 to +1 for x values within the portion (i.e., such that |x-Pcx| <|Psx|) whereas the image, in the calling subpicture's coordinate system, should only range from -1/2 to +1/2.)

If the image size is specified instead of the magnification, we have the following:

```
/_ |x |y_/ = (/_ x y_/ - /_Pcx Pcy_/) * / 1/Psx  0    / *
                                        /_  0  1/Psy _/

         / cosA sinA    / * / |sx 0    / + /_ |cx |cy_/
        /_ -sinA cosA _/   /_ 0  |sy _/
```

or, in other words,

```
2)
/_|x |y_/ = /_x-Pcx y-Pcy_/ * /|sx cosA/Psx |sy sinA/Psx /
                              /_-|sx sinA/Psy |sy cosA/Psy_/

           + /_|cx |cy_/
```

Expanding the parenthesized quantities in equations 1) and 2), we have:

```
3a) /_|x |y_/ = /_x y_/ * /Mx cosA/2Psx   Mx  sinA/2Psx /
                           /_-My sinA/2Psy  My cosA/2Psy_/

           + /_|cx-PcxMxcosA/2Psx+PcyMysinA/2Psy
                       |cy-PcxMxsinA/2Psx-PcyMycosA/2Psy _/
```

and

```
3b) /_|x |y_/ = /_x y_/ * /|sx cosA/Psx |sy sinA/Psx  /
                          /_-|sx sinA/Psy |sy cosA/Psy_/

           + /_|cx-Pcx|sxcosA/Psx+Pcy|sxsinA/Psy
                       |cy-Pcy|sysinA/Psx-Pcy|sycosA/Psy_/
```

Various interesting substitutions can be made in 3a) and 3b). For example, if A=0 (no rotation), then we have:

```
4a) /_|x |y_/ = /_x y_/ * /Mx/2Psx 0 / + /_|cx-PcxMx/2Psx
|cy-PcyMy/2Psy_/
                           /_ 0 My/2Psy_/

4b) /_|x |y_/ = /_x y_/ *  /|sx/Psx 0 / + /_|cx-Pcx|sx/Psx
|cy-Pcy|sy/Psy_/
                            /_ 0 |sy/Psy_/
```

Another example is if no portioning is done (Pcx=Pcy=0, Psx=Psy=1/2):

```
5a) /_|x |y_/ = /_ x y_/  * /Mx cosA Mx sinA   / + /_|cx |cy_/
                            /_-My sinA My sinA_/

5b) /_|x |y_/ = /_ x y_/ * /2|sx cosA 2|sy sinA   / + /_|cx |cy_/
                           /_-2|sx sinA 2|sx cosA_/
```

If in addition, 0=0, we have:

6a) /_|x |y_/ = /_ xMx+|cx yMy+|cy_/

6b) /_|x |y_/ = /_ x*2|sx+|cx y*2|sy+|cy_/

Of course, in all cases, the transformation from /_x y_/ to /_|x |y_/ can be written in the form:

```
/_|x |y_/ = /_x y_/ * / 2 by 2 / + /_ translation _/
                      /_ matrix _/
```

In general, a transformation combining a linear transformation and a translation is called an affine transformation.

Transformations with Nested Levels

The combination of two affine transformations is again an affine transformation. Indeed, if

```
/_|x |y_/ = /_x y_/ * / Mat1 / + /_ Tran1 _/
                      /_     _/
```

and

```
/_|x_ |y__/ = /_|x |y_/ * ( / Mat1 / * / Mat2 /)
                          /_    _/   /_    _/
            + (/_ Tran2 _/ + /_ Tran1 _/ * / Mat2 /)
                                        /_    _/
```

Thus if one has nested full subpicture calls, the data at any level need be transformed only once, namely, by the transformation which is the combination of the single step transformations at each level of nesting. A new "grand combination" affine transformation should be computed whenever a full subpicture is called (after pushing down the current transformation) by combining the current grand combination with the affine transformation for this particular subpicture call.

Portions with Nested Levels

As long as no rotations are involved, or even only rotations in multiples of pi/2, then multiple levels of portions are easy to implement. In the discussion in the next two paragraphs let us assume that no rotations other than whole multiples of pi/2 are involved.

Just as one can keep track of a "grand combination" affine transformation, so can one keep a grand combination of portions. At each level, one can proceed as follows: Save a copy of the current

grand portion, and use the inverse of the single level affine transformation (specified in the subpicture call) to determine what rectangle of the called page corresponds to the current grand portion (on calling page).

Various relations may exist between this rectangle and the rectangle specified (or defaulted) in the subpicture call. They may be disjoint (in which case this subpicture need not be called at all); they may be equal (an easy case); one may contain the other or they may partially overlap. If there is any intersection, it will be a rectangle, and this rectangle becomes the new grand combination portion.

The problem with rotations other than multiples of pi/2 is that the inverse image of the rectangle is no longer in the standard orientation (vertical and horizontal edges). This means that its intersection with the portion specified on the subpicture call may have 3, 4, 5, 6, 7, or 8 edges (if it is non-empty). Deeper levels may get even worse if they involve rotations too. While there may be no conceptual difficulty (for some) in working with such a situation, significantly more computation is involved than in the simple horizontal and vertical edge case.

This protocol puts forward no recommendation in the case that rotations other than whole multiples of pi/2 are involved with portions. It does suggest that nested portions be handled as described above in the more straightforward case.

[This RFC was put into machine readable form for entry]
[into the online RFC archives by Helene Morin, Via Genie, 12/1999]