Network Working Group
Request for Comment: 292
NIC 8302
References: 282, 285
Updates: None
Obsoletes: None

12 January 1972
Jim Michener, MAC
Ira Cotton, MITRE
Karl Kelley, U. of Ill.
Dave Liddle, Owens Ill.
Ed Meyer, MAC

GRAPHICS PROTOCOL - LEVEL 0 ONLY

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. In particular, only the simplest forms of graphics data will be covered in this, the first version of this document. The next version, already in preparation, will be much more complete. In any case this is not intended to describe a finished protocol; rather it should serve as a basis for graphics experimentation on 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 package 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 intergrated 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.

LEVEL 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 incremently. 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 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 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 an 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.

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 make his own decision, but it is recommended that the largest possible _square_ area be utilized.) 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 had no "normal" size, 72 characters per line would be desirable. Later, variable character size may be introduced.

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.

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 hosts and even programmable terminals should be able to implement it. In accordance with this, the "rule" was made that a command be implemented only if the 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>, <y>, <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> <y>
3	3	00000011	Move Relative <x> <y>
4	4	00000100	Draw Absolute <x> <y>
5	5	00000101	Draw Relative <x delta> <y delta>
6	8	00000110	Dot Absolute <x> <y>
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 ("Move Absolute") <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 ("Move Relative") <x delta> <y delta>.
Nothing is drawn; the beam is shifted by the specified amount in x and y.

4 Draw line to absolute position ("Draw Absolute") <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 ("Draw Relative") <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 ("Dot Absolute") <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 ("Dot Relative") <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 scopes(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.

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," and not "a <string> whose <count> is zero".

```
<graphics output byte stream> ::= empty_string
                    | <picture> <graphics output byte stream>
<picture> ::= <new picture stt> <sttgroup> <end stt>
<stt group> ::= empty_string | <stt> <stt group>.
<stt> ::= <control stt> | <display stt>
<control stt> ::= <escape to device stt>
                    | <null stt>
<display stt> ::= <move absolute stt>
                    | <move relative stt>
                    | <draw absolute stt>
                    | <draw relative stt>
                    | <dot absolute stt>
                    | <dot relative stt>
                    | <text and restore beam stt>
                    | <text stt>

<new picture stt> ::= ERASE
<escape to device stt> ::=ESCDEV <device code> <string>
<null stt>::= NULL
<end stt>::= ENDPIC
<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>
<text and restore beam stt> ::= TEXTR <string>
<text stt> ::= TEXT <string>
<x coordinate> ::= <coordinate>
<y coordinate> ::= <coordinate>
<x delta> ::= <double coordinate>
<y delta> ::= <double coordinate>
<coordinate> ::= singed,_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
```

```
<count ::= 7-bit_non-negative_integer
                  | 15-bit_non-negative_integer_represented_in
                  "excess_2**15"_notation
<string> ::= <count> count_8-bit_bytes
<device code> ::= <value>
<value> ::= 8-bit_integer
```