Network Working Group
Request for Comments: 553
NIC: 17810

C. Irby
K. Victor
SRI-ARC
14 July 1973

Draft design for a text/graphics protocol

DRAFT DESIGN FOR A TEXT/GRAPHICS PROTOCOL

This proposal should be seen as a synthesis of existing ideas rather than an attempt to put forth new ones. It is based on work by the NGG, Elaine Thomas, Peter Deutsch, Charles Irby, Ken Victor, Bill Duvall, Bob Sproull, and others at ARC, PARC, and BBN.

We are concerned about the lack of text-handling capabilities of the protocol suggested in RFC 493. Also, we feel that the protocol will have a significant influence on the interface provided to writers of future graphics application programs, and consequently that such things as "beam twiddling" should not be part of the protocol.

> Things of this nature address the problem at too low a level for a facility which is intended to service the needs of a wide range of graphics devices.
>
> We feel that, although the breakdown into "levels" as proposed in RFC 493 may be expedient for initial experimentation, it is inappropriate for a Network Standard Protocol. Instead, we propose that the protocol allow for two levels, segmented and structured. This allows the writers of graphics application programs to deal with a very simple display facility (segments consisting of lines, dots, or character strings) or with a powerful structure of display subroutines.

We propose an experimental implementation of such a scheme on the ARC, BBN, and PARC systems to test these ideas using several application programs (including NLS) and at least an IMLAC, ARDS, and an E&S LDS.

Environment

We are trying to design a protocol used to communicate with a "virtual display" to operate at the other end of a wire (ARPANET connection) from a "host" which is running some kind of display application program.

We will adopt the terminology that the human user, sitting at the display, is the "user" and the application program is the "server".

We wish to stress the fact that within a single application, a single terminal should be useable both as an "interactive graphics" terminal AND as an "interactive control" terminal. Thus, the graphics protocol must allow for teletype-like operations.

The need for two sets of connections for running graphics programs over the Net (according to our understanding) is centered about the issue of handling (being able to recover gracefully from) berserk programs (and perhaps achieving greater bandwidth through the net).

We recognize this problem but also think one should be able to run graphics programs using only one set of telnet connections. Also, it seems obvious that even though one is running a graphics program, one must expect to be able to handle "unescorted" characters (not embedded in a command or response message) being sent to his terminal.

Consequently, we are proposing that the graphics protocol be implemented within telnet using 8-bit BEGIN-GRAPHICS-COMMAND and END-GRAPHICS-COMMAND characters or the 8-bit transparent mode of the new telnet. This means that one will be able to run graphics programs with one, two, or more sets of telnet connections.

We also strongly propose that any site which is interested in supporting display terminals for use in network graphics would be prudent to implement local control over the display (such as IGNORE-GRAPHICS-COMMANDS, RESET-TO-TTY-MODE commands from the user to the using host). Failure to take such precautions may very well lead to burned out tubes!

Basic concepts

The model

The model we are adopting consists of an application program manipulating a (remote) display file. This file may be "segmented" or "structured", in which case it may be manipulated independently from the display itself.

For structured display files an "update display" command causes the display file to get mapped onto the display in whatever fashion is appropriate for the display.

Part of this protocol deals with commands issued to the (remote) display file editor. This editor creates and changes the display file at the user host.

Structured Display Files

A structured display file consists of named subpictures, each containing any number of named units. There are two types of units, primitive units and call units. The effect of a unit is independent of its name or creation within the subpicture.

Primitive units contain drawing instructions and associated coordinates that may generate visible information on the display screen. Drawing instructions and coordinates can occur only in primitive units.

Call units give the display structure a subroutine capability. A call unit invokes the display of another subpicture. In other words, a call unit allows one subpicture to contain instances of other subpictures. As well as providing for subroutine-style control transfer, call units can be used to establish display parameters and maintain parameter transparency. For example, a call unit can be used to call a subpicture with a translation and relative intensity setting. On return from the called subpicture, these parameters are restored to their original values.

A subpicture is an ordered list of units which can be any mixture of primitive and call units. Each subpicture begins with a header and terminates with the subpicture end unit. The subpicture end unit is a single unique unit in a display structure linked to the end of each subpicture.

In order to understand how control passes through a structure, one can think of the display elements as follows: subpictures are subroutines and units are linked blocks of in-line code. When all of the units contained in a subpicture have been executed, the subpicture end unit returns control to wherever the subpicture was called from. A primitive unit contains display code and transfer to the next unit. A call unit contains a subroutine call to a subpicture and a transfer to the next unit in line.

Segmented Display Files

A segmented display file consists of named segments, each containing any number of primitive units. The only operations available for segmented display file is to add new, delete old, or replace old segments (updating the actual display happens automatically). The effect of a unit is independent of its name or creation order within the subpicture.

Hosts

Since a given terminal may be under partial control of several different hosts, all further discussion of names, coordinates, display files, etc. should be taken as relative to each individual host.

That is, each host believes it has a display file, naming, and coordinate space and a set of state parameters entirely under its control; its only evidence of resource sharing is that the terminal may appear to be of different sizes at different times.

(We feel that in principle it should be processes within hosts, rather than hosts, that enjoy these properties, but it does not seem feasible to construct a process identification scheme that all hosts will find acceptable.)

Subpictures

A subpicture has a name and zero or more units.

Subpicture names are arbitrary, globally unique, fixed-length identifiers (subpicture names are chosen by the host).

Each unit (displayable component) has a name, which is local to the subpicture.

A unit may be a "primitive unit", such as a string or a vector, or a "call unit", which implies displaying a (possibly transformed) copy of another subpicture.

The display data are organized into a re-entrant tree (acyclic graph) by the call units.

A unit may be "visible" or "invisible".

A particular instance of a subpicture (the result of a call-unit) appears on the screen precisely if it and all subpictures on the logical path to it from the root of the tree are "visible".

Primitive units

Strings

A string is a horizontal line of characters with its own mode and X,Y origin relative to the origin of the subpicture.

Note: intensity is always relative.

Font and mode (e.g. blinking) information logically accompanies each character. This is accomplished by means of embedded mode and font specification characters and a "restore original string mode and font" character.

Note: Mode modifiers are non-displayable characters and do not take up character positions on the screen.

Determining the space occupied on the screen by a string requires knowledge of the font(s) being used; this is a separate question which is dealt with later.

TTY units

A tty unit is a rectangle that consists of a number of lines. Within this unit the display acts as if it were an alpha-numeric display, e.g.,

characters which would write beyond the right hand margin of the rectangle cause an automatic line folding to take place

ascii control characters CarriageReturn, LineFeed, FormFeed, and BackSpaceCharacter, (HorizontalTab and VerticalTab?), are interpreted appropriately

other control characters are displayed in a terminal specific manner, e.g. ^F, <^F>, etc.

display of the characters in the range 200-377 is left unspecified at this point (truncated to 7 bits?, alternate fonts?, alternate modes?)

It is hoped that we can agree on a standardization of some of the characters in this range to allow for such things as greek letters, common mathematical symbols, super-scripting, and sub-scripting.

linefolding that would cause characters to be written below the rectangle (whether performed automatically or by a LineFeed character, etc.) cause the text within the unit to be scrolled upwards one line (storage tube may adopt a different scheme).

Characters are displayed in a teletype unit in one of two ways:

Characters sent to the terminal that are not part of any command (unescorted characters) are appended to appropriate tty-units (see below --- USE-TTY-UNITS, TTY)

By use of the APPEND-STRING-TO-UNITS command for structured display files

The first character sent to a tty-unit appears as the first character (at the left margin) of the top line. This is necessary for a number of reasons, the most convincing of which is the behavior characteristics of storage tubes and most real alpha-numeric displays.

Successive characters appear as successive characters within the top line until either an explicit (e.g., linefeed) or implicit (line overflow) line break occurs.

When a line break occurs, the next character appears on the second from the top line of the unit.

This continues until the bottom line of the tty-unit is reached.

At this point, a line break causes the lines within the unit to scroll up one line.

Note: Storage scopes may use a different technique for scrolling.

Dots

A dot unit consists of an initial X0,Y0 followed by a series of points X,Y which describe a series of dots.

Each dot unit logically carries mode information such as blinking, relative intensity, etc.

Lines

A line unit consists of an origin X0,Y0 followed by a series of points X,Y which describes a series of straight lines connected tail-to-head (i.e. a polygon).

Each line unit logically carries mode information such as blinking, dotted vs. solid, invisible.

Other kinds of lines, such as conic sections, may belong in the primitive set.

Special points

This primitive unit consists of a series of points, which will be displayed joined by lines in the best available manner.

The intent is to use Flegal's algorithms to produce a smooth curve.

Device-specific

This primitive unit consists of any number of device specific commands. The device type may be obtained through an interrogation command.

Call units

In addition to the name of the referenced subpicture, a call unit may include the following transformations:

Master/instance rectangle: specifies a rectangle in the caller's space into which a specified rectangle of the callee's space is to be imaged. This provides independent scaling in each coordinate as well as translation and clipping.

Rotation. It may be desirable to combine this with scaling using the familiar idea of homogeneous transformation.

Intensity and color control. In principle, a call could specify intensity increments (positive or negative) for each color.

It is assumed that best effort will be used in scaling and rotation of text. We recommend replacing it by a line when all else fails.

Initial state

After the initial telnet connection is established, the first graphics command issued by the applications program should be a request for either a structured display file or for a segmented display file.

The response to this request should be whether or not the requested display file was allocated and other parameters about the virtual display, e.g. screen size, character sizes, whether or not color is available, etc.

Before the display file is allocated, the terminal should appear as, and simulate to the best of its ability, a Network Virtual Terminal (NVT).

Any graphic commands issued before the allocation of a display file will be ignored.

After requesting commands and receiving a structured display file, the following structure will exist:

There will exist a subpicture, referred to as the ICP SUBPICTURE, whose rectangular extent corresponds to the extent of the virtual display allocated to this host.

There will exist a tty-unit, referred to as the ICP TTY-UNIT, in the ICP SUBPICTURE, where rectangular extent corresponds to the extent of the virtual display allocated to this host.

This tty-unit will consist of n lines, where n is terminal dependent and available through a query command.

This tty-unit will be instituted for the display of unescorted characters.

There will be in effect an implicit call on the ICP SUBPICTURE.

This call is not accessible to the applications program.

The applications program causes the display of information by:

1) creating primitive units in the ICP SUBPICTURE

2) creating call units, to created subpictures, in the ICP SUBPICTURE

3) using the TTY command to make visible/invisible the ICP TTY-UNIT (or change its location or size)

After requesting and receiving a segmented display file, the following structure will exist:

There will exist a segment, referred to as the ICP SEGMENT.

There will exist a tty-unit, referred to as the ICP TTY-UNIT, in the ICP SEGMENT, whose rectangular extent corresponds to the extent of the virtual display allocated to this host.

This tty-unit will consist of n lines, where n is terminal dependent and available through a query command.

This tty-unit will be instituted for the display unescorted characters.

The applications program causes the display of information by:

1) creating primitive units in the ICP SEGMENT

2) creating new segments

3) using the TTY command to make visible/invisible the ICP TTY-UNIT (or to relocate it or change its size)

Display editing primitives

General editing primitives

REQUEST-DISPLAY-FILE (file-type)

file-type is either structured or segmented.

This command requires a response.

Segmented display file editing

SEGMENT (Segment)

If the segment Segment already exists, then it is cleared; if it did not exist then it is created.

Pictures are displayed within segments by the use of the primitive unit command listed below.

DELETE-SEGMENT(Segment)

If the segment exists, then it is deleted.

Primitive Units

All unit operations cause immediate display on the screen.

STRING-UNIT(Segment,Mode,X-Origin,Y-Origin,Text)

Writes the specified string unit.

Mode refers to relative intensity, blinking, reverse video, color, etc.

Errors: Segment does not exist.

LINE-UNIT(Segment,Type,Mode,X0,Y0,X1,Y1, ..., Xn,Yn)

Draws the specified line segments.

Type refers to solid, dashed, dotted, etc.

Errors: Segment does not exist; illegal mode.

DOT-UNIT(Segment,Mode,X0,Y0,X1,Y1, ..., Xn,Yn)

Draws the specified dots.

Errors: Segment does not exist; illegal mode.

SPECIAL-POINTS-UNIT(Segment,Mode,X1,Y1, ..., Xn,Yn)

Draws the special-points curve.

The terminal should attempt to connect the specified points in the nicest way possible (e.g. Flegal’s spline curve algorithm, straight line segments).

Errors: Segment does not exist; illegal mode.

TTY-UNIT(Segment,Mode,Rectangle,Lines)

Creates a unit which will behave as a tty-simulation area with "lines" lines distributed within the specified rectangle.

Unescorted characters will be echoed in this unit in addition to any other units they are being sent to.

Errors: Segment does not exist.

DEVICE-SPECIFIC-UNIT(Segment,device commands)

Creates a unit of device specific commands.

TTY(parameters)

parameters are:

position rectangle, visible/invisible, number of lines, mode of characters

This refers to the ICP TTY simulation.

RESET()

delete all segments, except ICP SEGMENT, and all units of ICP SEGMENT, except ICP TTY-UNIT

resets all nodes to their initial state (i.e., the state that existed immediately after a REQUEST-DISPLAY-FILE command)

Structured display file editing

SUBPICTURE(Subpicture, rectangle)

Creates a new subpicture with name "Subpicture". "rectangle" is the coordinates of a diagonal of the subpicture's virtual screen (i.e. its coordinate system.)

If a subpicture named "Subpicture" already exists, it is cleared and the new coordinate rectangle takes precedence.

DELETE-SUBPICTURE(Subpicture)

Deletes the subpicture named "Subpicture". Call units referring to Subpicture are also deleted.

CLEAR-SUBPICTURE(Subpicture)

Deletes all units of the subpicture Subpicture, but does not delete the subpicture.

Primitive Units

All the operations for creating units are transparent to the prior existence of the designated unit, i.e. they function as "replace" as well as "create".

STRING-UNIT(Subpicture,Unit,Target-Key,Mode,X-Origin,Y-origin,Text)

Replaces the unit by a string unit.

Mode specifies the mode of the characters (e.g. blinking, underlined, etc).

Target-Key is used in conjunction with the TARGET-SENSITIVE command and target input. It may also be sent via the SET-TARGET-KEY COMMAND.

Errors: Subpicture does not exist; X-Origin or Y-Origin is outside the subpicture's virtual coordinate system.

We explicitly do not require an error if the string extends beyond the right-hand edge of the subpicture; however, the results are not defined.

LINE-UNIT(Subpicture,Unit,Target-Key,Type,Mode,X0,Y0,X1,Y1, ..., Xn,Yn)

Replaces the unit by a line unit.

Errors: Subpicture does not exist illegal mode; some X or Y is outside the subpicture.

DOT-UNIT(Subpicture,Unit,Target-Key,Type,Mode,X0,Y0,X1,Y1, ..., Xn,Yn)

Replaces the unit by a dot unit.

Errors: Subpicture does not exist; illegal mode; some X or Y is outside the subpicture.

SPECIAL-POINTS-UNIT(Subpicture,Unit,Target-Key,Type,Mode,X1,Y1, ..., Xn,Yn)

Replaces the unit by a special-points unit.

Errors: Subpicture does not exist; illegal mode; some X or Y is outside the subpicture.

CALL-UNIT(Subpicture,Unit,Target-Key,Called-Subpicture,Parameters)

Replaces the unit by a call unit.

Parameters:

Master-Instance rectangles

rotation

mode

Errors: Subpicture does not exist; Called-Subpicture does not exist; parameter errors.

TTY-UNIT(Subpicture, unit, mode, rectangle, lines)

Creates a unit which will behave as a tty-simulation area with "lines" lines distributed within the specified rectangle.

Errors: Subpicture does not exist.

DEVICE-SPECIFIC-UNIT(Subpicture, Unit, Target-Key, device, commands)

Creates a unit of device specific commands. The action of the commands should leave alone (or at least restore) any global modes, e.g., the standout mode (see below).

APPEND-STRING-TO-UNIT(Subpicture, Unit, Text)

Appends the specified text to the specific commands. This only makes sense if the specified unit is a string or tty unit.

Errors: Subpicture does not exist, unit does not exist, not a string or tty unit.

DELETE-UNIT(Subpicture, Unit)

Deletes a unit.

VISIBLE-UNIT(Subpicture, Unit, Flag)

Makes the Unit visible or invisible as specified by Flag. If a unit which is target sensitive is made invisible, it is no longer target sensitive. However, in the absence of a subsequent modifying target sensitive command, the unit becomes target sensitive again if it should be made visible.

Errors: Subpicture does not exist, unit does not exist.

SET-TARGET-KEY(Subpicture, Unit, Target-Key)

Sets the target key for the specified unit to the specified value.

SET-STANDOUT-MODE(mode)

Sets the mode that will be used to make text and/or units stand out to blinking, underlining, etc.

If the terminal does not support the specified mode, the terminal should make a best effort or use another method to make things stand out.

STANDOUT-UNIT(Subpicture, unit, yesno)

makes the specified unit stand out (according to the mode set by SET-STANDOUT-MODE) or not, according to "yesno". If the unit which is to stand out is a call-unit, the instance of the subpicture which is the result of the call (all the way to the terminal nodes) is made to stand out.

STANDOUT-TEXT(Subpicture, unit, begin-char-count, end-char-count, yesno)

Unit must refer to a string unit.

Makes the specified text stand out (according to the mode set by SET-STANDOUT-MODE) or not, according to "yesno".

UPDATE-STRUCTURED-DISPLAY()

This causes any changes that have been made to the display file, since the last update or since ICP, to be reflected on the screen.

TTY(parameters)

parameters are:

position rectangle, visible/invisible, number of lines, mode of characters

This refers to the ICP TTY simulation

USE-TTY-UNITS(Subpicture1, unit1, ..., Subpicturen, unitn)

Unescorted characters are to be appended only to the specified tty units.

Errors: Subpicture, unit does not exist.

RESET(How)

Case How Of

= Permanent

Immediately resets the terminal to its initial ICP state

= Temporary

Immediately resets the terminal to its initial ICP state without destroying the previous state.

= Restore state saved from last RESET(Temporary).

Direct Feedback

It seems extremely desirable, given network speeds, to allow the using host to perform direct feedback to the user without intervention from the application program in the serving host. This is already done in telnet with local echoing. We propose extending this capability to graphics by allowing "dragging" (attaching a subpicture's origin to the position of the cursor), "tracking" (following the movement of the mouse, stylus, or light pen with a distinctive mark on the screen), "inking" (plotting the trail of the cursor on the screen) and "rubber banding" (a straight line attached to a fixed point on one end the cursor location on the other).

These should be seen as allowable extensions of the protocol rather than as requirements. There should, however, be commands available in the protocol for determining their existence and controlling them.

Data input primitives

Input Control

TARGET-SENSATIVE(key1, ..., keyn)

Arms the units which have the specified keys for target selection.

SET-INPUT-MODE(Device, parameters)

Selects the mode in which a logical device shall produce input and under what conditions.

the logical devices are specified below as well as their possible input formats and conditions.

Errors: no such device.

Keyboard input

The keyboard has only one input mode, in which it sends a character whenever a key is struck.

Binary devices

Unless otherwise specified, binary devices act as an extension of the keyboard and produce 8-bit characters which are not distinguishable from keyboard characters by the serving host.

The algorithm for translating binary devices into characters is not specified, but something like the NLS accumulation algorithm for mouse-keyset chords is intended.

Binary devices may also input binary data (according to their up/down states), which is transmitted on state changes. Examples of this type of device are function keys and overlay cards, mouse and keyset (used independently or together), pen-up/pen/down, light pen buttons, etc.

Coordinate input

Coordinates may be sent according to any subset of the following criteria: with every character in some designated set (e.g. control characters, or all characters); with every binary device state change input; after some time interval has elapsed; after a position change P > (y1-y0) ^2+(x1-x0)^2, etc.

Coordinates may be sent in either or both of "X-Y" or "target" format.

X-Y format is just the location of the cursor relative to the screen region assigned to the host.

Target format is the "call stack" (logical path from the root unit - the ICP SUBPICTURE - to the closest unit) plus the target-key of that unit plus the count of the closest character within the string or the closest line segment or dot or special point if appropriate.

Target input is unavailable for segmented display files.

In the event of overlapping target sensitive units, it is not specified which of the units selected will be returned as the hit unit.

Time input

Since hosts may wish to consider two events happening sufficiently close together to be simultaneous, or to keep detailed interaction statistics, it must be possible to request time information to be sent with some reasonable subnet of other types of input.

Interrogations

It must be possible for the serving host to discover its environment (e.g. screen size, available devices) and to read back state information (display file).

This is very desirable both for debugging and for redirecting a displayed image to another device (e.g. a plotter).

Environment

Terminal parameters: screen size and resolution, available input devices, terminal type (for device specific control), number of lines in the ICP TTY-UNIT.

Character parameters: available character sizes, special (non-ASCII) characters, font characteristics, sub- and super-scripting facilities.

State

Display file or display file components.

Cursor Position

It should be possible for the application program to read the cursor position at any time.

Display File Support

It should be possible to find out if this user process supports only segmented or structured display files, or both.

Command support

It should be possible to get a matrix from the user process which indicates which commands are implemented. This is a necessity to find out which, if any, of the direct feedback features are supported, and might be nice to allow for, e.g., the possibility of a text only or graphics only subset of the protocol to be implemented.

Encoding Principles

Commands will have the format : BGC OPCODE DATA EGC where:

BGC (Begin Graphics Command) places the telnet connection into a "read graphics command" mode,

OPCODE DATA is the specific graphics command and data, and

EGC (End Graphics Command) restores the telnet connection to its normal state.

Note: This may all have to be bracketed by telnet Begin-8-bit-transparent-mode and End-8-bit-transparent-mode commands.

Numbers in general will have have 7-bits of significance in each byte -- if the high order of a byte is on, then the significant bits from the next byte should be concatenated onto the low-order end of the bits collected so far, etc..

Subpicture names - shall be 14-bit numbers, assigned by the serving host.

Unit names - shall be 14-bit numbers, assigned by the serving host.

Strings - shall be 8-bit characters, with an escape convention to represent changes of font and mode.

Since the channel is 8-bits wide, there is room for many more than 128 displayable characters. However, the interpretation of codes 200B and above is not standardized!

Coordinates should be as described in RFC 493.

Rectangles - shall be specified by the coordinates of the endpoints of one of the diagonal.

Encoding

The actual encoding of this protocol is forthcoming. Since we expect some changes to come about because of the upcoming Network Graphics Group Meeting, we have postponed the actual encoding until after this meeting.

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