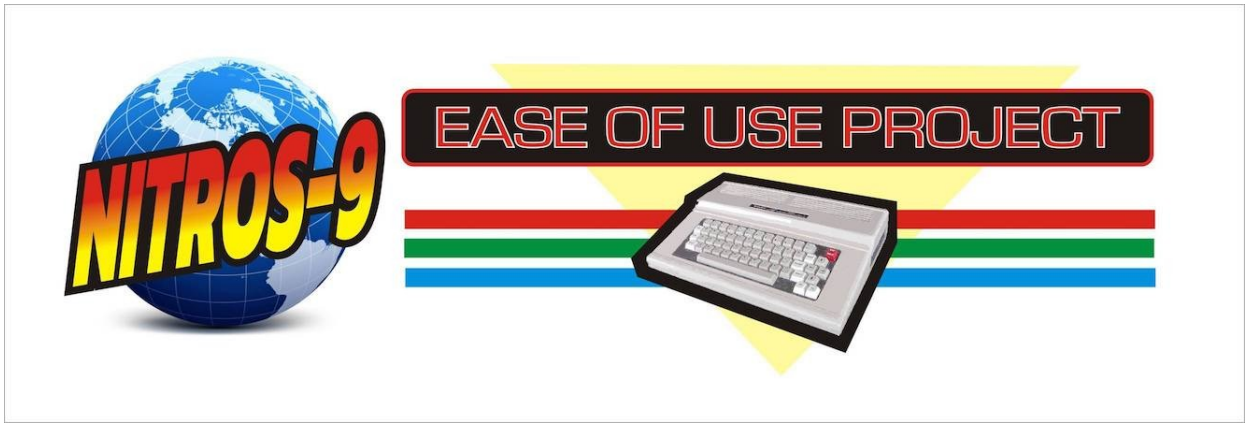


NitrOS-9 EOU Level 2 Windowing System Manual

The NitrOS-9 EOU Project
http://www.lcurtisboyle.com/nitros9/nitros9.html

The NitrOS-9 project:
http://sourceforge.net/projects/nitros9/

Revision History:

Revision	**Date**	**Comment**
0.30	December 3,2020	Corrections, additions for EOU Beta 6
0.21	July 2, 2020	Minor additions about optimizations.
0.2	May 19, 2020	Updated to reflect changes to NitrOS-9 over time and specifically to cover changes related to the EOU (“Ease of Use”) project.
0.1	June 4, 2004	Created

Acknowledgements:

2004 revision by: Bob Emery and Boisy Pitre.

2020 revision additions and corrections by: L. Curtis Boyle and Jay Searle

Table Of Contents

.....1
NitrOS-9 EOU Level 2 Windowing System Manual.....1
Table Of Contents.....3
Chapter 1. The NitrOS-9 Level 2 Windowing System.....5
Device Windows.....6
Opening a Device Window.....6
Overlay Windows.....9
Opening an Overlay Window.....9
Chapter 2. Overview of Commands and Parameters.....10
Parameters.....10
Chapter 3. General Commands.....13
Bcolor.....14
BoldSw.....15
Border.....16
CWArea.....17
DefColr.....18
DfnGPBuf.....19
DWEnd.....23
DWProtSw.....24
DWSet.....25
FColor.....27
Font.....28
GCSet.....31
GetBlk.....32
GPLoad.....33
KilBuf.....34
LSet.....35
OWEnd.....37
OWSet.....38
Palette.....40
PropSw.....42
PSet.....43
PutBlk.....46
ScaleSw.....47
Select.....48
TCharSw.....49
Chapter 4. Drawing Commands.....50
Arc3P.....51
Bar.....52
RBar.....52
Relative Draw Bar.....52
Box.....53
RBox.....53
Circle.....54
Ellipse.....55
FFill.....56
FCircle.....58
FEllipse.....59
Line.....60

Rline....60
LineM....61
RLineM....61
Point....62
RPoint....62
PutGC....63
SetDPtr....64
RSetDPtr....64
Chapter 5. Text Commands....65
Optimization Tips:....66
Alphabetical Index....67

Chapter 1. The NitrOS-9 Level 2 Windowing System

One of NitrOS-9 Level 2's advanced features is its built-in Windowing System. The Windowing System allows you to lay one or more rectangular areas, called *windows*, on top of your existing screen display. These smaller areas can take up a portion of the screen, or utilize an entire screen area.

With these windows, you can watch several tasks perform at the same time. For example, suppose you are writing a business letter using a word processor in one window. You can go to a spreadsheet program in another window, get a price quote you need, return to the word processor, and include the price in the letter. In yet another window, a terminal program connected to a modem could be downloading a program or data file.

There are a number of options that windows give to your application and work environments: they can be text only or contain a combination of text and graphics. Separate text, drawing and background colors can be selected as well to provide a particular color style. The ability to customize your window to suite your working environment and preferences is easy to do and puts the power of windowing at your fingertips.

The Windowing System allows as many windows as your computer's memory can support, with a maximum of 32 available at one time, including overlay windows.

Support for the windowing system comes from several modules:

- CoGrf – Handles the parsing of display codes for window creation and manipulation.
- CoWin – Performs the same functions as CoGrf, but also adds in enhanced window border and mouse control functionality (use only CoGrf or CoWin in your system, not both!)
- GrfDrv – This module is located in the CMDS directory of your boot disk and is responsible for carrying out the actual drawing and bitmap manipulation functions. It is automatically loaded as needed by NitrOS-9.

In NitrOS-9, there are two types of windows: device windows and overlay windows.

Device Windows

A *device window* is one that can run a program or utility. This is the type of window you would use in the word processor/spreadsheet example given above. Each device window acts as an individual terminal.

The device windows are designated as devices /w1 - /w15. You open a device window as you do any other NitrOS-9 device and specify the window's parameters, including whether the window is for text or graphics. If you want to run a process in the window, you can start an execution environment, such as a *shell*, on the window. (See "Opening a Device Window," later in this chapter, and the DWSet command in Chapter 3)

> **Note:** If you want only to send output to the device window without running a process in the window, do not start a shell on the window.

Device windows cannot overlay each other, and their boundaries cannot overlap.

Opening a Device Window

To open a device window, follow these steps:

1. First, we must allocate memory for the window. Use NitrOS-9's iniz command to initialize window 7 in this example. Type:

   `iniz /w7` ENTER

2. Next, send an escape sequence to the window that tells it the parameters you want. These parameters include the screen type, size, and colors. For example:

   `wcreate /w7 -s=2 20 10 40 10 01 00` ENTER
   OR
   `display 1b 20 2 14 0a 28 0a 01 00 00>`ENTER

sends the DWSet command escape sequence to the /w7 window. The **wcreate** command insists that you use decimal numbers, while the **display** command can take both decimal and hexadecimal numbers.

The functions of the codes, as used in the **wcreate** command, are as follows:

2	Sets a screen type of 80 x 24 (text only)
20	Starts the window at character/column 20
10	Starts the window at line/row 10
40	Sets a window size of 40 characters
10	Sets a window height of 10 lines
01	Sets the foreground color to blue
00	Sets the background color to white
00	Sets the border color to white

If you do not send escape sequences, NitrOS-9 uses default descriptors for the windows.

The window size defaults are:

Window Number	**Screen Type (chars/line)**	**Starting Position (horizontal, vertical)**	**Size (columns, rows)**
Term	40 (text)	0,0	40,25
1	80 (text)	0,0	80,25
2	80 (text)	0,0	80,25
3	80 (text)	0,0	80,25

4	80 (text)	0,0	80,25
5	80 (text)	0,0	80,25
6	80 (text)	0,0	80,25
7	80 (text)	0,0	80,25
8	80 (text)	0,0	80,25
9	80 (text)	0,0	80,25
10	80 (text)	0,0	80,25
11	80 (text)	0,0	80,25
12	80 (text)	0,0	80,25
13	80 (text)	0,0	80,25
14	80 (text)	0,0	80,25
15	80 (text)	0,0	80,25

3. Use NitrOS-9's **shell** command to fork a shell to the window. Type:

`shell i=/w7&` ENTER

The **i=** parameter creates an immortal shell. Creating an immortal shell protects the window and its shell from being destroyed if you accidentally exit the shell using CTRL BREAK.

You now have a window that can run its own tasks. Information displayed in that window is automatically scaled to the window's size.

Overlay Windows

An *overlay window* is a window that you open on top of a device window. (You can place overlay windows over other overlay windows, but there must always be a device window at the bottom of the stack.) The purpose of overlay windows is to display computer dialog. You cannot fork a shell to an overlay window; however, you can run a shell in an overlay window. Overlay windows assume the screen type of the device windows they overlay.

Opening an Overlay Window

To open an overlay window, use the Overlay Window Set function.
(See OWSet in Chapter 3, “General Commands”)

Chapter 2. Overview of Commands and Parameters

The windowing commands are divided among three chapters, based on their functions.

Chapter 3 describes the *general commands*. These commands let you create windows and buffers, access buffers, set switches, and maintain the window environment.

Chapter 4 describes the *drawing commands*. Besides letting you draw all kinds of images (circles, ellipses, arcs, and boxes, to name a few), these commands also enable you to color areas or to fill them with patterns.

Chapter 5 describes the *text commands*. Use these commands to manipulate the text cursor and the text attributes. Text commands operate on hardware text screens (screen types 1 and 2) and graphics windows if a font is selected.

Each command description lists the command's name, code, and parameters. To call a Windowing System command using NitrOS-9's **display** command, type **display**, followed by the command code and the values you want to supply for the parameters.

Parameters

The following is a complete list of the parameter abbreviations used in Chapters 3, 4, and 5. All parameters represent a single byte of information.

Parameter	**Description**
HBX	*high order byte of x value*
LBX	*low order byte of x value*
HBY	*high order byte of y value*
LBX	*low order byte of y value*
HBXo	*high order byte of x-offset value (relative)*
LBXo	*low order byte of x-offset value (relative)*
HBYo	*high order byte of y-offset value (relative)*
LBYo	*low order byte of y-offset value (relative)*
HBR	*high order byte of radius*
LBR	*low order byte of radius*

HBL	*high order byte of length*
LBL	*low order byte of length*
HSX	*high order byte of size in x direction*
LSX	*low order byte of size in x direction*
HSY	*high order byte of size in y direction*
LSY	*low order byte of size in y direction*
HBRx	*high order byte of radius in x direction*
LBRx	*low order byte of radius in x direction*
GRP	*GET/PUT buffer group number (1-254)*
BFN	*GET/PUT buffer number (1-255)*
LCN	*logic code number*
PRN	*palette register number (0-15, wraps mod 16)*
CTN	*color table number (0-63, wraps mod 64)*
FNM	*font number*
CPX	*character position x (0-xmax)*
CPY	*character position y (0-ymax)*
STY	*screen type*
SVS	*save switch (0 = no save, 1 = save area under overlay)*
SZX	*size in x (columns)*
SZY	*size in y (rows)*
XDR	*dimension ratio x used with YDR as YDR/XDR*
YDR	*dimension ratio y*
BSW	*binary switch (0 = off, 1 = on)*

Chapter 3. General Commands

The general commands let you set up and customize windows. They also let you set up and access image buffers and select colors for the screen.

Bcolor

Background Color

Function: Lets you choose a color palette register to use as the background color. See the Palette command for setting up the actual colors.

Code: 1B 33

Parameters: PRN

BoldSw

Bold Switch

Function: Enables or disables boldfacing for text on graphics screens. If boldface is on, the screen displays subsequent characters in bold. If boldface is off, the screen displays subsequent characters in the regular font.

Code: 1B 3D

Parameters: BSW

BSW = switch
00 = off (Default)
01 = on

Notes:

- You can use BoldSw with any font.
- Boldface is not supported on hardware text screens (screen types 1 and 2).

Border

Screen Color

Function: Lets you change the palette register used for the screen border. See the Palette command for setting up the actual colors.

Code: 1B 34

Parameters: PRN

Notes:

- You set the border by selecting a palette register to use for the border register. When the actual color is changed in the palette register selected by the command, the color of the screen border changes to the new color. In general, the border register usually matches the background palette register.

CWArea

Change Working Area

Function: Lets you alter the working area of the window. Normally, the system uses this call for high-level windowing, but you can use it to restrict output to a smaller area of the window.

Code: 1B 25

Parameters: CPX CPY SZX SZY

Notes:

- You cannot change a window's working area to be larger than the predefined-area of the window as set by DWSet or OWSet.
- All drawing and window updating commands are done on the *current working area* of a window. The working area defaults to the entire size of the window. Scaling, when in use, is also performed relative to the current working area of a window. The CWArea command allows users to restrict the working area of a window to smaller than the full window size. Functions that might be performed by opening a non-saved overlay window to draw or clear an image and then closing the overlay can be accomplished by using this command to shorten execution time where an actual overlay window is not needed.

DefColr

Default Color

Function: Sets the palette registers back to their default values. The actual values of the palette registers depend on the type of monitor you are using. (See **montype** in *NitrOS-9 Commands Reference.)*

Code: 1B 30

Parameters: None

Notes:

- The default color definitions apply only to high-resolution graphics and text displays.
- The system sets the palette registers to a proper compatibility mode when switching to screens using the older VDG emulation modes. See the table below:

Window System Color Modes		**VDG-Compatible Modes**	
Palette	*Color*	*P# Color*	*P# Color*
00 & 08	White	00 Green	08 Black
01 & 09	Blue	01 Yellow	09 Green
02 & 10	Black	02 Blue	0A Black
03 & 11	Green	03 Red	0B Buff
04 & 12	Red	04 Buff	0C Black
05 & 13	Yellow	05 Cyan	0D Green
06 & 14	Magenta	06 Magenta	0E Black
07 & 15	Cyan	07 Orange	0F Orange

- The SetStat call SS.DFPal lets you change the default color palette definition when using the windowing system. Default colors in the VDG-Compatible Mode cannot be changed. See the *NitrOS-9 Technical Reference* manual for information on SetStat.

- The system's default colors are used whenever you create a new window.

DfnGPBuf

Define GET/PUT Buffer

Function: Lets you define the size of the GET/PUT buffers for the system. Once you allocate a GET/PUT buffer, it remains allocated until you use the KilBuf command to delete it.

NitrOS-9 allocates memory for GET/PUT buffers in 8K blocks that are then divided into the different GET/PUT buffers. Buffers are divided into buffer groups. Therefore, all commands dealing with GET/PUT buffers must specify both a group number and a buffer number within that group.

Code: 1B 29

Parameters: GRP BFN HBL LBL

Technical:
The buffer usage map is as follows:

Group Number	Buffer Number[1]	Use
0	1-255	Internal use only (returns errors)
1-199	1-255	General use by applications[2]
200-254	1-255	Reserved
255	1-255	Internal use only for overlay windows (returns errors)

Note: The names, buffer groups, and buffer numbers are defined in the assembly definition file. The decimal number you use to call these are in parentheses next to the name.

[1] Buffer Number 0 is invalid and cannot be used.
[2] The application program should request its user ID via the GETID system call to use as its group number for buffer allocation.

For example, to select the Arrow pointer, Grp_Ptr and Ptr_Arr, you use 202,1 as the group/buffer number.

The standard group numbers are defined as follows:

Grp_Fnt(200) = font group for system fonts
- Fnt_S8x8(1) = standard 8x8 font
- Fnt_S6x8(2) = standard 6x8 font
- Fnt_G8x8(3) = standard graphics font

A complete list of current fonts can be found in dd/sys/fontlist.txt

The standard fonts are in the file SYS/StdFonts.

Grp_Clip(201) = clipboarding group (for Multi-Vue, MVCanvas, etc.)
Grp_Ptr(202) = graphics cursor (pointer) group
- Ptr_Arr(1) = arrow pointer (hp = 0,0)
- Ptr-Pen(2) = pencil pointer (hp = 0,0)
- Ptr_LCH(3) = large cross hair pointer (hp = 7,7)
- Ptr_Slp(4) = sleep indicator (hourglass)
- Ptr_Ill(5) = illegal indicator
- Ptr-Txt(6) = text pointer (hp = 3,3)
- Ptr_SCH(7) = small cross hair pointer (hp = 3,3)
- hp=hit point, the coordinates of the actual point on the object at which the cursor should be centered.

The standard pointers are in the file SYS/StdPtrs.

Grp_Pat2(203) = two color patterns
Grp_Pat4(204) = four color patterns
Grp_Pat6(205) = sixteen color patterns
- Pat_Dot(1) = dot pattern
- Pat_Vrt(2) = vertical line pattern
- Pat_Hrz(3) = horizontal line pattern
- Pat_XHtc(4) = cross hatch pattern

Pat_LSnt(5) = left slanted lines
Pat_RSnt(6) = right slanted lines
Pat_SDot(7) = small dot pattern
Pat_BDot(8) = large dot pattern

NEW TO BETA 6 – EXTENDED PATTERNS (USED IN MVCANVAS)

Pat.Bubl(9) = bubble pattern
Pat.Bskt(10) = basket weave pattern
Pat.Wafl(11) = waffle iron pattern
Pat.Shng(12) = shingle pattern
Pat.Bric(13) = brick pattern
Pat.SmGr(14) = small grid pattern
Pat.Dmnd(15) = Diamond floor pattern
Pat.Flor(16) = floor tile pattern

Each pattern is found within each of the pattern groups.
Standard patterns are in the files:
SYS/StdPats_2, SYS/StdPats_4, and SYS/StdPats_16.
Extended patterns are in the files:
SYS/StdPats_2.plus, SYS/StdPats_4.plus, and SYS/StdPats_16.plus.

Grp.Wnd(206) = Multi-Vue / Gshell Extended 4 color 3D look characters
Wnd.UpAr(1) = Up arrow 3D look
Wnd.DnAr(2) = Down arrow 3D look
Wnd.LtAr(3) = Left arrow 3D look
Wnd.RtAr(3) = Right arrow 3D look
Wnd.VBar(3) = Vertical scroll bar marker 3D look
Wnd.HBar(3) = Horizontal scroll bar marker 3D look

Grp.Cd16(207) = 16 color playing card set (courtesy of Paul Shoemaker)
These are buffer numbers 1-54:
1=Card back
2-14=(Ace to King of Diamonds)
15-27=(Ace to King of Hearts)
28-40=(Ace to King of Spades)
41-53=(Ace to King of Clubs)
54=Joker
55=Empty card slot

NOTE: **EOU Beta 6 and up** now also pre-load the **MVCanvas** extended patterns. The 16 color card playing set is NOT pre-loaded, but available for use in SYS/CARDDECK. Each card image has it's own buffer, so you only need to load the specific cards that you need for a particular card game, if you wish.

All files have GPLoad commands imbedded in file, along with the data. To load fonts, pointers, or patterns, simply merge them to any window device: For example:

```
merge SYS/StdFonts ENTER
```

sends the standard font to standard output which may be redirected to another device if the current output device is not a window device (such as when **term** is a VDG screen).

You only need to load fonts once for the entire system. Once a GET/PUT buffer is loaded, it is available to all devices and processes in the system.

NitrOS-9 EOU Beta 2 and up specific:

By default, the EOU preloads all fonts for you, so you have access to all of them in your programs.

DWEnd

Device Window End

Function: Ends a current device window. DWEnd closes the display window. If the window was the last device window on the screen, DWEnd also deallocates the memory used by the window. If the window is an interactive window, NitrOS-9 automatically switches you to a new device window, if one is available.

Code: 1B 24

Parameters: None

Notes:

- DWEnd is only needed for windows that have been attached via the **iniz** utility or the I$Attach system call. Non-attached windows have an implied DWEnd command that is executed when you close the path.

DWProtSw

Device Window Protect Switch

Function: Disables and enables device window protection. By default, device windows are protected so that you cannot overlay them with other device windows. This type of protection helps avoid the possibility of destroying the contents of either or both windows.

Code: 1B 36

Parameters: BSW

BSW = switch
00=off
01 = on (Default)

Notes:

- We recommend that you not turn off device window protection. If you do, however, use extreme discretion because you might destroy the contents of the windows. NitrOS-9 does not return an error if you request that a new window be placed over an area of the screen which is already in use by an unprotected window.

- **GShell** uses this command to create an underlying, non-protected device window underneath it's resizable protected device windows, to allow you to position multiple device windows on the same screen, and you can use this technique as well.

DWSet

Device Window Set

Function: Lets you define a window's size and location on the physical screen. Use DWSet after opening a path to a device window.

Code: 1B 20

Parameters: STY CPX CPY SZX SZY PRN1 PRN2 PRN3
PRN 1 = Foreground
PRN2 = Background
PRN3 = Border (if STY >= 1)

Notes:

- The **iniz** and **display** commands open paths to the device window.
- When using DWSet in a program, you must first open the device. If you are changing the window type of your current window, you must issue a DWEnd first.
- Output to a new window is ignored until NitrOS-9 receives a DWSet command, unless defaults are present in the device descriptor (/w1-/w15). If defaults are present in the device descriptors, NitrOS-9 automatically executes DWSet, using those defaults.
- When NitrOS-9 receives the DWSet command, it allocates memory for the window, and clears the window to the current background color. If the standard font is already in memory, NitrOS-9 assigns it as the default font. If the standard font is not in memory, you must execute a font set (Font) command after loading the fonts to produce text output on a graphics window. If you do not have a font loaded, all characters will show as periods '.'
- Use the Screen Type code (STY) to define the resolution and color mode of the new screen. If the screen type code is zero, NitrOS-9 opens the window on the process's currently selected screen. If the code is 255 ($FF), NitrOS-9 opens the window on the currently displayed screen. If the code is non-zero, NitrOS-9 allocates a new screen for the window.

The following describes the acceptable screen types:

Code	Screen Size	Colors	Memory	Type
255	Current Displayed Screen[3]			
00	Process's Current Screen			
01	40 x 25	8 & 8	2000	Text
02	80 x 25	8 & 8	4000	Text
05	640 x 200	2	16000	Graphics
06	320 x 200	4	16000	Graphics
07	640 x 200	4	32000	Graphics
08	320 x 200	16	32000	Graphics

Note: The GIME has a bug and will only show 199 lines in modes 05-08, but the 200th line is there and will show up when scrolling upwards.

- The location of the window on the physical screen is determined by the diagonal line defined by:

  (CPX,CPY) and (CPX + SZX, CPY + SZY)

- The foreground, background, and border register numbers (PRN1, PRN2, and PRN3) define the palette registers used for the foreground and background colors. See the Palette command in this chapter for more information.
- When an implicit or explicit DWSet command is done on a window, the window automatically clears to the background color.
- All windows on the screen must be of the same type (either text or graphics).
- Values in the palette register affect all windows on the screen. However, you can choose which register to use for foreground and background for each window. That is, NitrOS-9 maintains palette registers and border register numbers for the entire screen and foreground and background registers numbers for each individual window.

[3] Use the Current Displayed Screen option only in procedure files to display several windows on the same physical screen. All programs should operate on that process' current screen.

- NitrOS-9 deallocates memory for a screen when you terminate the last window on that screen.

FColor

Foreground Color

Function: Lets you select a color palette register for the foreground color.
See the Palette command for setting the actual colors.

Code: 1B 32

Parameters: PRN

Font

Select Font

Function: Lets you select/change the current font. Before you can use this command, the font must be loaded into the specified GET/PUT group and buffer (using GPLoad). See the GPLoad command for information on loading font buffers.

Code: 1B 3A

Parameters: GRP BFN

Notes:

- You can select proportional spacing for the font by using PropSw.
- All font data is a 2-color bit map of the font.
- Each character in the font data consists of 8 bytes of data. The first byte defines the top scan line, the second byte defines the second scan line, and so on. The high-order bit of each byte defines the first pixel of the scan line, the next bit defines the next pixel, and so on. For example, the letter "A" would be represented like this:

Byte	Pixel Representation
$10	. . . # . . . .
$28	. . # . # . . .
$44	. # . . . # . .
$44	. # . . . # . .
$7C	. # # # # # . .
$44	. # . . . # . .
$44	. # . . . # . .
$00	. . . .

Note that 6x8 fonts ignore the last 2 bits per byte.

- For fonts with 128 characters or less, the fonts are ordered with characters in the following ranges:

$00-$1F International characters (see mapping below)
$20-$7F Standard ASCII characters

International characters or any characters in the font below character $20 (hex) are printed according to the following table:

Character position in font	Char1 or	Char2
$00	$C1	$E1
$01	$C2	$E2
$02	$C3	$E3
$03	$C4	$E4
$04	$C5	$E5
$05	$C6	$E6
$06	$C7	$E7
$07	$C8	$E8
$08	$C9	$E9
$09	$CA	$EA
$0A	$CB	$EB
$0B	$CC	$EC
$0C	$CD	$ED
$0D	$CE	$EE
$0E	$CF	$EF
$0F	$D0	$F0
$10	$D1	$F1
$11	$D2	$F2
$12	$D3	$F3
$13	$D4	$F4
$14	$D5	$F5
$15	$D6	$F6
$16	$D7	$F7
$17	$D8	$F8
$18	$D9	$F9
$19	$DA	$FA
$1A	$AA	$BA
$1B	$AB	$BB
$1C	$AC	$BC
$1D	$AD	$BD

$1E	$AE	$BE
$1F	$AF	$BF

For 224 character fonts, Character positions $00 to $1F in the font map to character codes $E0 to $FF, and character positions $20 to $DF are the same character code as their position.

NOTE: When you select a new font, the X position of the text cursor is returned to 0 (as if a CR has occurred), and then the font takes effect.

GCSet

Graphics Cursor Set

Function: Creates a GET/PUT buffer for defining the graphics cursor that the system displays. You must use GCSet to display a graphic cursor.

Code: 1B 39

Parameters: GRP BFN

Notes:

- To turn off the graphics cursor, specify GRP as 00.
- A system standard buffer or a user-defined buffer can be used for the graphics cursor.
- If you specify a buffer # that doesn't exist, GCSet leaves it at the previously working buffer #.

GetBlk

Get Block

Function: Saves an area of the screen to a GET/PUT buffer. Once the block is saved, you can put it back in its original location or in another on the screen, using the PutBlk command.

Code: 1B 2C

Parameters: GRP BFN HBX LBX HBY LBY HSX LSX HSY LSY
HBX/LBX = *x-location of block (upper left corner)*
HBY/LBX = *y-location of block*
HSX/LSX = *x-dimension of block*
HSY/LSY = y-dimension *of block*

Notes:

- The GET/PUT buffer maintains information on the size of the block stored in the buffer so that the PutBlk command works more automatically.
- If the GET/PUT buffer is not already defined, GetBlk creates it. If the buffer is defined, the data must be equal to or smaller than the original size of the buffer.
- Original OS-9 Level 2 had a limit of not being able to do a GetBlk that was the full width of the screen. This is fixed in NitrOS-9.

> ***NitrOS-9 EOU Beta 4 and up specific:***
>
> You are now allowed to use GetBlk/PutBlk with hardware text screens. When doing this, you specify the X,Y start positions and sizes by character, not pixel. You can also get and put between windows, but the window types and sizes currently MUST be the same type and size. It will also currently wrap on the right hand side to one line lower on the left side.

- It is recommended, for your own programs, to use the group number that corresponds to your process ID, so that you don't overwrite another programs buffers by accident. Since some older programs do not clean up buffers themselves, you

may also want to do a **KilBuf** for your group number with buffer 0 (Kill all get/put buffers) at the start of your program.

GPLoad

GET/PUT Buffer Load

Function: Preloads GET/PUT buffers with images that you can move to the screen later, using PutBlk.

If the GET/PUT buffer is not already created, GPLoad creates it.

If the buffer was previously created, the size of the passed data must be equal to or smaller than the original size of the buffer. Otherwise, GPLoad truncates the data to the size of the buffer.

Code: 1B 2B

Parameters: GRP BFN STY HSX LSX HSY LSY HBL LBL (Data…)
STY = format
HSX/LSX = x-dimension *of stored block*
HSY/LSY = y-dimension *of stored block*
HBL/LBL = *number of bytes in data*

Notes:

- Buffers are maintained in a linked list system.
- Buffers to be used most should be allocated last to minimize the search time in finding the buffers.
- When loading a Font GET/PUT Buffer, the parameters are as follows:

GRP BFN STY HSX LSX HSY LSY HBL LBL (Data…)
GRP = 200 (note: not enforced, but HIGHLY recommended)
STY = 5
HSX/LSX = *x-dimension size of Font (6 or 8)*
HSY/LSY = *y-dimension size of Font (8)*
HBL/LBL = *size of first data* (not including this header information)

See the Font command for more information on font data.

KilBuf

Kill GET/PUT Buffer

Function: Deallocates the buffer specified by the group and buffer number. To deallocate the entire group of buffers, set the buffer number to 0.

Code: 1B 2A

Parameters: GRP BFN

Notes:

- KilBuf returns memory used by the buffer to a free list. When an entire block of memory has been put on the free list, the block is returned to the system.
- It is recommended that you use the same group number as your process ID number, so that you don't change get/put buffers in other processes. It is also recommended that you Kill the entire group at the start of your program, since some older programs did not clean up buffers after they were exited.

LSet

Logic Set

Function: Lets you create special effects by specifying the type of logic used when storing data, which represents an image, to memory. The specified logic code is used by all draw commands until you either choose a new logic or turn off the logic operation. To turn off the logic function, set the logic code to 00.

Code: 1B 2F

Parameters: LCN

LCN = logic *code number*
- 00 = No logic code; store new data on screen
- 01 = AND new data with data on screen
- 02 = OR new data with data on screen
- 03 = XOR new data with data on screen

Notes:

- The following tables summarize logic operations in bit manipulations:

AND	First Operand	Second Operand	Result
	1	1	1
	1	0	0
	0	1	0
	0	0	0

OR	First Operand	Second Operand	Result
	1	1	1
	1	0	1
	0	1	1
	0	0	0

XOR	First Operand	Second Operand	Result
	1	1	0
	1	0	1
	0	1	1
	0	0	0

- Data items are represented as palette register numbers in memory. Since logic is performed on the palette register number and not the colors in the registers, you should choose colors for palette registers carefully so that you obtain the desired results. You may want to choose the colors for the palette registers so that LSet appears to *and*, *or*, and *xor* the colors rather than the register numbers. For example:

Palette #	**Color**	**Alternative Order**
0	White	Black
1	Blue	Blue
2	Black	Green
3	Green	White

OWEnd

Overlay Window End

Function: Ends a current overlay window. OWEnd closes the overlay window and deallocates memory used by the window. If you opened the window with a *save switch* value of hexadecimal 01, NitrOS-9 restores the area under the window. If you did not, NitrOS-9 does not restore the area and any further output is sent to the next lower overlay window or to the device window, if no overlay window exists.

Code: 1B 23

Parameters: None

OWSet

Overlay Window Set

Function: Use OWSet to create an overlay window on an existing device window. NitrOS-9 reconfigures current device window paths to use a new area of the screen as the current logical device window.

Code: 1B 22

Parameters: SVS CPX CPY SZX SZY PRN1 PRN2

SVS = save switch

00 = Do not save area overlayed , nor clear it.

01 = Save area overlayed and restore at close

PRN1 = *foreground palette register*

PRN2 =*background palette register*

Notes:

- If you set SVS to zero, any writes to the new overlay window destroy the area under the window. You might want to set SVS to zero if your system is already using most of its available memory. You might also set SVS to zero whenever it takes relatively little time to redraw the area under the overlay window once it is closed.
- If you have ample memory, specify SVS as 1. Doing this causes the system to save the area under the new overlay window. The system restores the area when you terminate the new overlay window. (See OWEnd.)
- The size of the overlay window is specified in standard characters. Use the same resolution (number of characters) as the device window that will reside beneath the overlay window. Have your program determine the original size of the device window at startup (using the SS.ScSiz GETSTAT call), if the device window does not cover the entire screen. See the *NitrOS-9 Technical Reference* manual for information on the SS.ScSiz GETSTAT call.
- Overlay windows can be created on top of other overlay windows; however, you can only write to the top most window. Overlay windows are "stacked" on top of each other logically. To get back down to a given overlay, you must close (OWEnd) any overlay windows that reside on top of the desired overlay window.

- Stacked overlay windows do not need to reside directly on top of underlying overlay windows. However, all overlay windows must reside within the boundaries of the underlying device window.

Palette

Change Palette

Function: Lets you change the color associated with each of the 16 palette registers.

Code: 1B 31

Parameters: PRN CTN

Notes:

- Changing a palette register value causes all areas of the screen using that palette register to change to the new color. In addition, if the border is set to that palette register, the border color also changes. See the Border command for more information.
- Colors are made up by setting the red, green, and blue bits in the color byte which is inserted in the palette register. The bits are laid out as follows:

Bit	Color
0	Blue low
1	Green low
2	Red low
3	Blue high
4	Green high
5	Red high
6	unused
7	unused

By using six bits for color (2 each for red, green and blue) there is a possibility of 64 from which to choose. Some of the colors are defined as shown:

White : 00111111 = $3F (all color bits set)
Black : 00000000 = $00 (no color bits set)
Standard Blue : 00001001 = $09 (both blue bits set)
Standard Green : 00010010 = $12 (both green bits set)
Standard Red : 00100100 = $24 (both red bits set)

- These colors are for RGB monitors. The composite monitors use a different color coding and do not directly match pure RGB colors. To get composite color from the RGB colors, the system uses conversion tables. The colors were assigned to match the RGB colors as close as possible. There are, however, a wider range of composite colors, so the colors without direct matches were assigned to the closest possible match. The white, black, standard green, and standard orange are the same in both RGB and composite.

PropSw

Proportional Switch

Function: Enables and disables the automatic proportional spacing of characters. Normally, characters are not proportionally spaced.

Code: 1B 3F

Parameters: BSW

BSW = switch
00 = off (Default)
01 = on

Notes:

- Any standard software font used in a graphics screen can be proportionally spaced.
- Proportional spacing is not supported on hardware text screens.

PSet

Pattern Set

Function: Selects a preloaded GET/PUT buffer as a pattern RAM array.
This pattern is used with all draw commands until you either change the pattern or turn it off by passing a parameter of 00 as GRP (Group Number).

Code: 1B 2E

Parameters: GRP BFN

Notes:

- The pattern array is a 32 x 8 pixel representation of graphics memory. The color mode defines the number of bits per pixel and pixels per byte. So, be sure to take the current color mode into consideration when creating a pattern array.
- The GET/PUT buffer can be of any size, but only the number of bytes as described by the following table are used:

Color Mode	Size of Pattern Array
2	4 bytes x 8 = 32 bytes (1 bit per pixel)
4	8 bytes x 8 = 64 bytes (2 bits per pixel)
16	16 bytes x 8 = 128 bytes (4 bits per pixel)

- The buffer must contain at least the number of bytes required by the current color mode. If the buffer is larger than required, the extra bytes are ignored.
- To turn off patterning, set GRP to 00.
- The following example creates a two color pattern of vertical lines. A two color pattern is made up of 1's and 0's. The diagram below shows the bit set pattern (note that one pixel is equal to one bit):

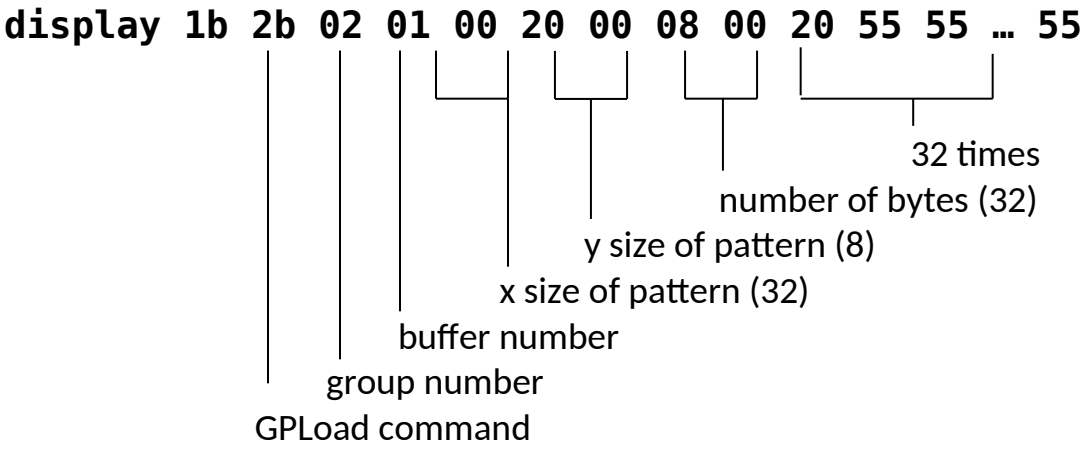
```
10101010101010101010101010101010
10101010101010101010101010101010
10101010101010101010101010101010
10101010101010101010101010101010
10101010101010101010101010101010
```

```
10101010101010101010101010101010
10101010101010101010101010101010
10101010101010101010101010101010
```

When the binary for the 2x8 pixel data is compressed into byte data, notice that each row consists of four bytes of data. The pattern now looks like this:

```
$55 $55 $55 $55
$55 $55 $55 $55
$55 $55 $55 $55
$55 $55 $55 $55          $55 = 01010101
$55 $55 $55 $55
$55 $55 $55 $55
$55 $55 $55 $55
$55 $55 $55 $55
```

To load the pattern in the system, use the GPLoad command. To load this particular pattern into Group 2 and Buffer 1, the command would be:



- When making a pattern using four colors, a pixel is made up of two bits instead of one. This means that the pattern consists of 64 bytes instead of 32. The diagram below shows the bit set pattern for the same vertical pattern using 4 colors:

```
1100110011001100110011001100110011001100110011001100110011001100
1100110011001100110011001100110011001100110011001100110011001100
1100110011001100110011001100110011001100110011001100110011001100
1100110011001100110011001100110011001100110011001100110011001100
1100110011001100110011001100110011001100110011001100110011001100
```

```
11001100110011001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100110011001100
11001100110011001100110011001100110011001100110011001100110011001100
```

When the binary for the 4x8 pixel data is compressed into byte data, notice that each row consists of 8 bytes of data. The pattern now looks like this:

```
$CC $CC $CC $CC $CC $CC $CC $CC
$CC $CC $CC $CC $CC $CC $CC $CC
$CC $CC $CC $CC $CC $CC $CC $CC
$CC $CC $CC $CC $CC $CC $CC $CC          $CC = 11001100
$CC $CC $CC $CC $CC $CC $CC $CC
$CC $CC $CC $CC $CC $CC $CC $CC
$CC $CC $CC $CC $CC $CC $CC $CC
$CC $CC $CC $CC $CC $CC $CC $CC
```

To load the pattern in the system, use the GPLoad command as described for the 2 color example, but specify $40 (64) bytes instead of $20 (32).

- When making a pattern using 16 colors, a pixel is made up of four bits instead of one. This means that the pattern consists of 128 bytes. Each line in the bit pattern would look like this:

  11110000…(repeat pattern for 16 total sets)…11110000

  When the binary for the 8x8 pixel data is compressed into byte data, the pattern is a series of $F0.

  To load the pattern in the system, use the GPLoad command and specify $80 (128) bytes as the size.

- Please see the **FFill** section for some special notes that pertain to it when using patterns.

PutBlk

Put Block

Function: Moves a GET/PUT buffer, previously copied from the screen or loaded with GPLoad, to an area of the screen.

Code: 1B 2D

Parameters: GRP BFN HBX LBX HBY LBY
HBX/LBX = *x-location of block* (upper left corner)
HBY/LBY = *y-location of block*

Notes:

- The dimensions of the block were saved in the GET/PUT buffer when you created it. NitrOS-9 uses these dimensions when restoring the buffer.
- The screen type conversion is automatically handled by the PutBlk routine in the driver. Please note that PutBlk will slow down noticeably if screen type conversion is required.
- GET/PUT buffers cannot be scaled. The image will be clipped if it does not fit within the window.
- If PutBlk's X coordinates are byte aligned with the original GetBlk, the PutBlk is much faster.

ScaleSw

Scale Switch

Function: Disables and enables automatic scaling. Normally, automatic scaling is enabled. When scaling is enabled, coordinates refer to a relative location in a window that is proportionate to the size of the window. When scaling is disabled, coordinates passed to a command will be the actual coordinates for that type of screen relative to the origin of the window.

Code: 1B 35

Parameters: BSW

BSW = *switch*
00 = off
01 = on (Default)

Notes:

- A useful application of disabled scaling is the arrangement of references between a figure and text.
- All coordinates are relative to the window's origin (0,0). The valid range for the coordinates:

  Scaling enabled:
  y = 0-199 for 25 line screens
  0-191 for 24 lines or less screens
  x = 0-639

  Scaling disabled:
  y = 0-size of y - 1
  x = 0-size of x - 1

Select

Window Select

Function: Causes the current process's window to become the active (display) window. You can select a different window by using the form:

```
display 1B 21>/wnumber
```

where *number* is the desired window number. If the process that executes the select is running on the current interactive (input/display) window, the selected window becomes the interactive window, and the other window becomes passive.

Code: 1B 21

Parameters: None

Notes:

- The keyboard is attached to the process's selected window through the use of the CLEAR key. This lets you input data from the keyboard to different windows by using the CLEAR key to select the window. All display windows that occupy the same screen are also displayed.
- The device window that owns the keyboard is the current interactive window. The interactive window is always the window being displayed. Only one process may receive input from the keyboard at a time. Many processes may be changing the output information on their own windows; however, you can only see the information that is displayed on the interactive window and any other window on the same screen as the interactive window.

TCharSw

Transparent Character Switch

Function: Defines the character mode to be used when putting characters on the graphics screens. In the default mode (transparent off), the system uses block characters that draw the entire foreground and background for that cell.

When in transparent mode, the only pixels that are changed are the ones where the character actually has pixels set in its font. When transparent mode is off, all pixels in the character block are set: foreground or font pixels in the foreground color and others in the background color.

> ***NitrOS-9 EOU Beta 2 and up specific:***
>
> Now works on hardware text screens. This will allow you to change the foreground color, underline and blink attributes while leaving the background color alone.

Code: 1B 3C

Parameters: BSW

BSW = switch

00 = off (Default)

01 = on

Chapter 4. Drawing Commands

All drawing commands relate to an invisible point of reference on the screen called the draw *pointer*. Originally, the draw pointer is at position 0,0. You can change the position by using the SetDPtr and RSetDPtr commands described in this chapter. In addition, some draw commands automatically update the draw pointer.

For example, the LineM command draws a line from the current draw pointer position to the specified end coordinates and moves the draw pointer to those end coordinates. The Line command draws a line but does not move the pointer. Also, note that all draw commands are affected by the pattern and logic commands described in Chapter 3

Do not confuse the draw pointer with the graphics cursor. The graphics cursor is the graphic representation of the mouse/joystick position on the screen.

An optimization note on all the Line commands – horizontal and vertical lines draw much faster than diagonal lines.

An optimization note when using fonts: Since fonts are natively stored as two color bit-maps (one bit per pixel), the corresponding graphical screen type (Type 5) will render text the fastest. If you do not require multiple colors, but want either multiple fonts and/or single color graphics on the screen simultaneously, this mode will render text much faster (and take less RAM).

In this chapter, commands that use relative coordinates *(offsets)* are listed with their counterparts that use absolute coordinates. For example, RSetDPtr is listed with Set-DPtr.

Arc3P

Draw Arc

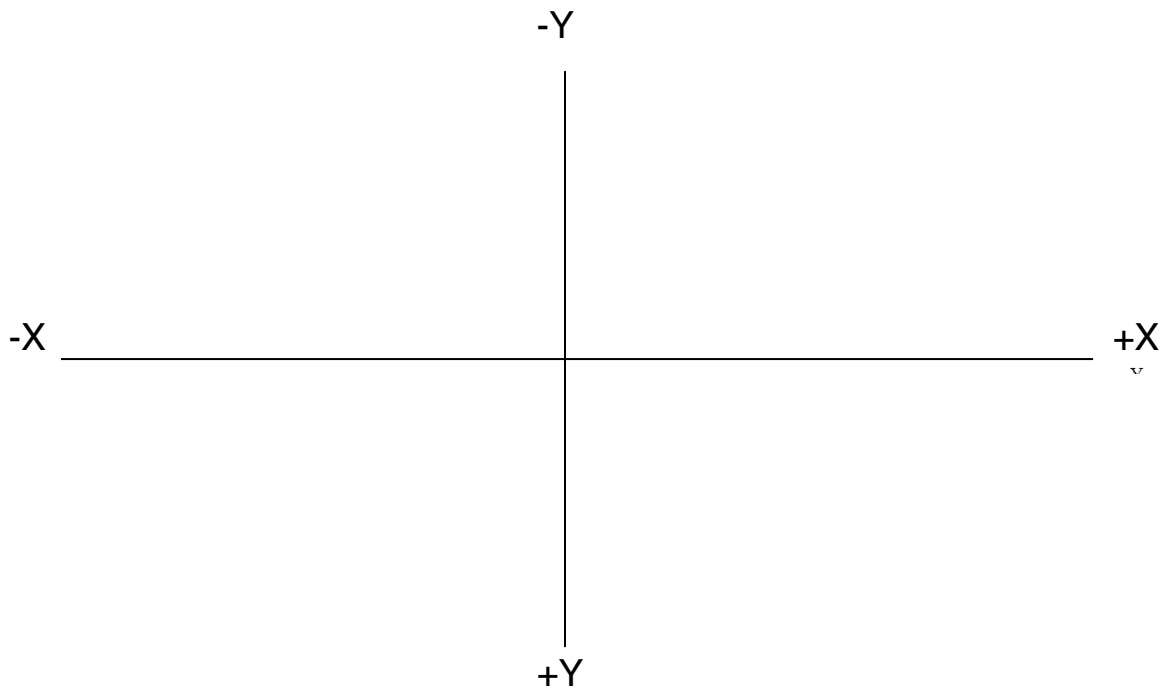
Function: Draws an arc with its midpoint at the current draw pointer position. You specify the curve by both the X and Y dimensions, as you do an ellipse. In this way, you can draw either elliptical or circular arcs. The arc is clipped by a line defined by the (X01,Y01) - (X02,Y02) coordinates. These coordinates are signed 16 bit values and are relative to the center of the ellipse. The draw pointer remains in its original position.

Code: 1B 52

Parameters: HBRx LBRx HBRy LBRy HX01 LX01 HY01 LY01 HX02 LX02 HY02 LY02

Notes:

- The resulting arc depends on the order in which you specify the line coordinates. Arc3P first draws the line from Point 1 to Point 2 and then draws the ellipse in a clockwise direction.
- The coordinates of the screen are as follows:



Bar

Draw Bar

Function: Draws and fills a rectangle that is defined by the diagonal line from the current draw pointer position to the specified position. The box is drawn in the current foreground color. The draw pointer returns to its original location.

Code: 1B 4A

Parameters: HBX LBX HBY LBY

RBar

Relative Draw Bar

Function: Draws and fills a rectangle that is defined by the diagonal line from the current draw pointer to the point specified by the offsets. The box is drawn in the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 4B

Parameters: HBXo LBXo HBYo LBYo

Box

Draw Box

Function: Draws a rectangle that is defined by the diagonal line from the current draw pointer position to the specified position. The box is drawn in the current foreground color. The draw pointer returns to its original location.

Code: 1B 48

Parameters: HBX LBX HBY LBY

RBox

Relative Draw Box

Function: Draws a rectangle that is defined by the diagonal line from the current draw pointer point specified by the offsets. The box is drawn in the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 49

Parameters: HBXo LBXo HBYo LBYo

Circle

Draw Circle

Function: Draws a circle of the specified radius with the center of the circle at the current draw pointer position. The circle is drawn in the current foreground color. The draw pointer remains in its original location.

Code: 1B 50

Parameters: HBR LBR

Ellipse

Draw Ellipse

Function: Draws an ellipse with its center at the current draw pointer position. The X value specifies the horizontal radius, and the Y value specifies the vertical radius. The ellipse is drawn in the current foreground color. The draw pointer remains in its original location. This is a relative command.

Code: 1B 51

Parameters: HBRx LBRx HBRy LBRy

FFill

Flood Fill

Function: Fills the area where the background is the same color as the draw pointer. Filling starts at the current draw pointer position, using the current foreground color. The draw pointer returns to its original location. This is a relative command.

Code: 1B 4F

Parameters: None

Notes: Patterns will generally work well with any commands that support them (like **Bar**, etc), but FFill is one that you have to be careful with. Because **FFill** uses the stack to keep track of every single point it has to come back to carry on filling (when it hits a different color), and there is not a ton of stack space in the **Grfdrv** system map, excessively complex fill areas can cause a stack overflow error. To compound this further, because you are changing the pixel values in a non-consistent way, it has the potential to get stuck in an infinite loop as it never returns to the initial start point to finish. We have put in special code to force it to exit if it seems to be stuck in such a loop, or gets trapped without reaching every part it should have, and there is also special code to allow **FFill** to try to "finish" as many paint points as it can even if a stack overflow occurs (using the points already on the stack). The net result of all of this is that it should never completely lock up the graphics driver subsystem (which previous versions could), and also that the results may not be perfect on very complex, large paint areas. As a programmer/developer, to avoid such issues, some guidelines/tips to note:

1) Try and make sure that **FFill**'s with patterns are always in small confined areas. If you want a whole background to be a pattern, with a bunch of smaller objects/drawing done as well, do a Bar with pattern first instead, and THEN draw your additional objects/shapes. I have encountered no issues with small, or larger non-complex, shapes to FFill in.

2) Add an error trap so that you can handle the Stack Overflow error, if you can't fix things with point 1) above. Please note that sometimes it may complete the **FFill** cor-

rectly even with the error; it depends on what points were saved and what their surroundings were like. But sometimes it won't complete everything properly, either.
3) Use other drawing commands with Patterns instead. All others (B**ar, Box, Filled Circle, Filled Ellipse,** etc.) are fixed areas of the screen, and thus don't potentially "leak" into other areas. **FFill** is the only command that can, so it should be used only when absolutely necessary.

4) **FFill**'s with plain colors and no patterns can still potentially create a stack overflow error if it is a hugely complex paint area, but it is much rarer than with a pattern (once an area is filled, it becomes "off limits" to the boundary checking, and thus doesn't loop back trying to repaint areas that it has already done).

FCircle

Draw filled circle

Function: Draws a filled circle of the specified radius with the center of the circle at the current draw pointer position. The filled circle is drawn in the current foreground color. The draw pointer remains in its original location.

Code: 1B 53

Parameters: HBR LBR

FEllipse

Draw Filled Ellipse

Function: Draws an filled ellipse with its center at the current draw pointer position. The X value specifies the horizontal radius, and the Y value specifies the vertical radius. The filled ellipse is drawn in the current foreground color. The draw pointer remains in its original location. This is a relative command.

Code: 1B 54

Parameters: HBRx LBRx HBRy LBRy

Line

Draw Line

Function: Draws a line from the current draw pointer position to the specified point, using the current foreground color. The draw pointer returns to its original location.
NOTE: Vertical and Horizontal lines draw much faster than diagonal ones.

Code: 1B 44

Parameters: HBX LBX HBY LBY

Rline

Relative Draw Line

Function: Draws a line from the current draw pointer position to the point specified by the x,y offsets, using the current foreground color. The draw pointer returns to its original location. This is a relative command.
NOTE: Vertical and Horizontal lines draw much faster than diagonal ones.

Code: 1B 45

Parameters: HBXo LBXo HBYo LBYo

LineM

Draw Line and Move

Function: Draws a line from the current draw pointer position to the specified point, using the current foreground color. The draw pointer stays at the new location.
NOTE: Vertical and Horizontal lines draw much faster than diagonal ones.

Code: 1B 46

Parameters: HBX LBX HBY LBY

RLineM

Relative Draw Line and Move

Function: Draws a line from the current draw pointer position to the point specified by the offsets, using the current foreground color. The draw pointer stays at the new location. This is a relative command.
NOTE: Vertical and Horizontal lines draw much faster than diagonal ones.

Code: 1B 47

Parameters: HBXo LBXo HBYo LBYo

Point

Draw Point

Function: Draws a pixel at the specified coordinates, using the current foreground color.

Code: 1B 42

Parameters: HBX LBX HBY LBY

RPoint

Relative Draw Point

Function: Draws a pixel at the location specified by the offsets, using the current foreground color. This is a relative command.

Code: 1B 43

Parameters: HBXo LBXo HBYo LBYo

PutGC

Put Graphics Cursor

Function: Puts and displays the graphics cursor at the specified location. The coordinates passed to this command are not window relative. The horizontal range is 0 to 639. The vertical range is 0 to 191 or 199 depending on whether the window is 24 lines or 25 lines. The default position is 0,0.

This command is useful for applications running under CoGrf (or CoWin) so that you can display a graphics cursor even if you don't want mouse control of the cursor.

Code: 1B 4E

Parameters: HBX LBX HBY LBY

SetDPtr

Set Draw Pointer

Function: Sets the draw pointer to the specified coordinates. The new draw pointer position is used as the beginning point in the next draw command if other coordinates are not specified.

Code: 1B 40

Parameters: HBX LBX HBY LBY

RSetDPtr

Relative Set Draw Pointer

Function: Sets the draw pointer to the specified in the offsets. The new draw pointer position is used as the beginning point in the next draw command if other coordinates are not specified. This is a relative command.

Code: 1B 41

Parameters: HBXo LBXo HBYo LBYo

Chapter 5. Text Commands

The text commands let you control the cursor's position and movement and also the way text prints on the display. These commands can be used on either text or graphics windows.

Code	Description
01	Homes the cursor.
02	Positions cursor to X/Y. Specify coordinates as (x + $20) and (y + $20).
03	Erases the current line.
04	Erases from the current character to the end of the line.
05 20	Turns off the cursor.
05 21	Turns on the cursor.
06	Moves the cursor right one character.
07	Rings the bell.
08	Moves the cursor left one character.
09	Moves the cursor up one line.
0A	Moves the cursor down one line.
0B	Erases from the current character to the end of the screen.
0C	Erases the entire screen and homes the cursor.
0D	Sends a carriage return.
1F 20	Turns on reverse video
1F 21	Turns off reverse video
1F 22	Turns on underlining.
1F 23	Turns off underlining.
1F 24	Turns on blinking.
1F 25	Turns off blinking.[4]
1F 30	Inserts a line at the current cursor position.
1F 31	Deletes the current line.
1B 3C BSW	See TCharSw[5]
1B 3D BSW	See BoldSw in Chapter 3[6]
1B 3F BSW	See PropSw in Chapter 3[6]

[4] Blink is not supported for text on graphics screens.

[5] See TCharSw description for how this works between hardware text and graphics screens.

[6] These characteristics are supported for text on graphics screens only.

Optimization Tips:

Some notes on how to get the best performance from your programs that are using the **Grfdrv** based graphics system:

1) Window scrolling, **GetBlk/PutBlk**'s on byte aligned boundaries, horizontal lines, boxes, bars and Flood fills have been optimized so that larger widths of these will run faster than previous versions (including **EOU Beta 5**). The absolute optimum condition for the greatest gain in speed is on even 8 byte boundaries (so every 4 characters on a hardware text screen, and every 16 pixels on 16 color screens, every 32 pixels on 4 color screens, and every 64 pixels on 2 color screens), and in even multiples of 8 bytes in width. But anything >=8 bytes wide should be faster than before, and <8 bytes wide is only marginally slower.

2) For scrolling in particular, windows that are the full width of the screen will scroll slightly faster than windows that are not the full width of the screen, irregardless of height. This is more noticeable on the **6309** versions of NitrOS-9, but also affects the **6809** versions as well.

3) 8 pixel wide fonts are optimized to run faster than 6 pixel wide fonts, or when the **proportiona**l attribute is turned on. Like up to several times faster.

4) Using a font that is pre-bolded in the font itself, it slightly faster at printing than picking a normal font and turning the **Bold** attribute on.

5) **PutBlk's** that end up running off the bottom of the window are optimized to not slow down (actually, they speed up), which is good for certain effects. We plan on expanding this functionality to the other 3 sides of a window in the future, but through **EOU Beta 6**, it is currently only optimized on the bottom.

Alphabetical Index

Arc3P....51
Bar....52
BoldSw....15, 65
Border....16, 25, 40
box....52f.
Box....53
circle....54, 58
Circle....54
CWArea....17
DefColr....18
device window....6, 9, 23ff., 37ff., 48
Device Window....6, 23ff.
DfnGPBuf....19
DWEnd....23
DWProtSw....24
DWSet....6f., 17, 25f.
ellipse....51, 55, 59
Ellipse....55, 59
FCircle....58
FColor....27
FFill....56
fill10
Fill....56
font....10f., 15, 20, 22, 25, 28ff., 33, 42, 49
Font....25, 28, 33
GCSet....31
GetBlk....32
GPLoad....21, 28, 33, 44ff.
GrfDrv....5
GrfInt....5, 63
KilBuf....19, 32, 34
line....7, 20, 26, 28, 45, 47, 50ff., 60f., 65
Line....50, 60f.
LineM....50, 61
LSet....35f.
overlay window....9, 17, 37f.
Overlay Window....9, 37f.
OWEnd....37f.
OWSet....9, 17, 38
Palette....14, 16, 18, 26f., 36, 40
Point....51, 62
PropSw....28, 42, 65

PSet....43
PutBlk....32f., 46
PutGC....63
RBar....52
RBox....53
Relative Draw Box....53
Relative Draw Line....60
Relative Draw Line and Move....61
Relative Draw Point....62
Relative Set Draw Pointer....64
Rline....60
RLineM....61
RPoint....62
RSetDPtr....50, 64
ScaleSw....47
Select....28, 48
SetDPtr....50, 64
system....5, 17ff., 22f., 31, 33f., 38, 41, 44f., 49
System....1, 5, 10, 18
TCharSw....49
WindInt....5
27