QuickDraw II
External Reference Specification

Steven E. Glass
Bennet Marks

August 11, 1986

Revision History

First Draft June 15, 1985	S. Glass
Revised July 10, 1985	S. Glass
Revised July 25, 1985	S. Glass
Revised September 19, 1985	S. Glass
Revised September 25, 1985	S. Glass
Revised October 29, 1985	S. Glass
Revised November 21, 1985	S. Glass
Revised December 3, 1985	S. Glass
Revised January 15, 1986	S. Glass
Revised March 5, 1986	S. Glass & B. Marks
Revised April 4, 1986	S. Glass & B. Marks
Revised April 25, 1986	S. Glass & B. Marks
Revised July 15, 1986	S. Glass & B. Marks
Revised August 11, 1986	S. Glass & B. Marks

Summary

People writing software want to be able to do Macintosh-like graphics in the Super Hi-Res Graphics modes.[1] To do this we need QuickDraw like routines for the new modes. Most of the code that provides this facility is in ROM; other parts will be in RAM (Because of the way the tools system works an application need not care what part of the code is where.). The ROM routines combined with these RAM based extensions are what we call **QuickDraw II.**

Appendix C contains a comparison of QuickDraw with QuickDraw II.

QuickDraw II Capabilities

The Quick Draw II core routines include calls for manipulating the graphics environment and drawing primitive graphical objects. Included in the graphics environment is information about

Drawing Location
Coordinate System
Clipping

The primitive objects supported are

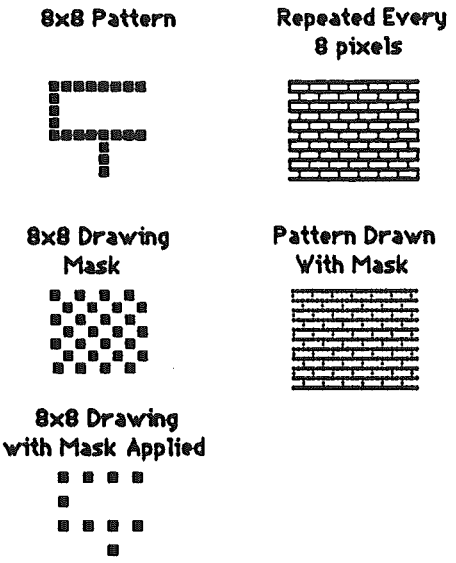
Lines
Rectangles
Regions
Polygons
Ovals
RoundRects
Arcs

Pixel Images
Text

The first group of objects (lines, rectangles, regions, polygons, ovals, roundrects and arcs) are all drawn with patterns. A pattern is a 64-pixel image organized as an 8x8 pixel square which is used to define a repeating design. When a pattern is drawn, it is aligned such that adjacent areas of the same pattern in the same graphics port will blend with it into a continuous, coordinated pattern. In addition to the pattern, lines, regions and rectangles are drawn using a drawing mask. The drawing mask is an 8x8 bit square that represents a repeating design that is used to mask the pattern as it is drawn. Only those pixels in the pattern aligned with a "1" in the mask are drawn. See figure 0 below.

[1]These are new modes found only on a Cortland computer and have no relationship to existing Apple II modes. The hardware is summarized in Appendix B for those not familiar with it.

Figure 0
Drawing with Patterns and Masks.



Note that drawing with a mask that is all "1" is like drawing without masking occuring at all. Drawing with a mask of all "0" is like not drawing since all pixels are masked out.

The QuickDraw II support for Pixel Maps is similar to Macintosh QuickDraw support for bit maps. The major difference is that pixels are not a single bit. An additional difference is that code in ROM does not support stretching and compression of an image as it is transfered from one to pixel map to another.

The QuickDraw II support for Text is similar to the text support on the Macintosh but not identical. Major differences include

QuickDraw II only supports a limited number of style modifcations
The Font Manager is not closely integrated with QuickDraw II. It is more of a higher level tool so the interaction between the two is different.
QuickDraw II does not scale text.

Basic Concepts and Terminology

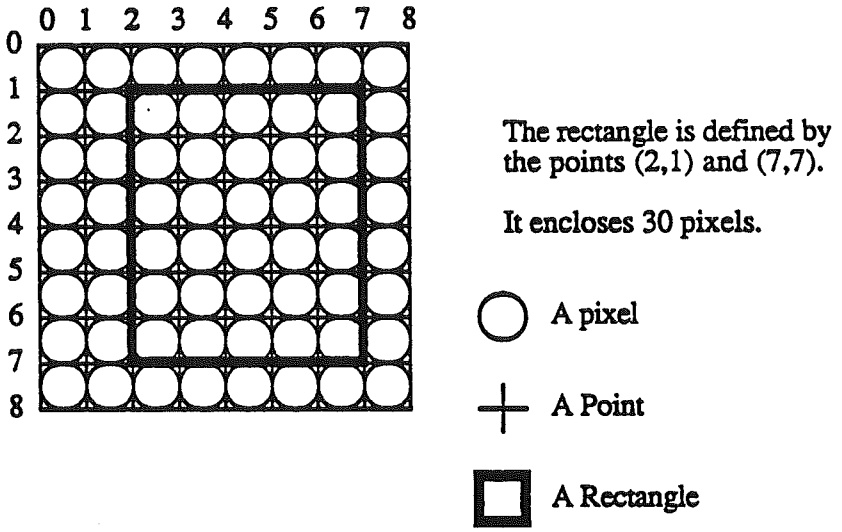
A pixel map is an area of memory containing a graphic image (the analogous QuickDraw term is BitImage). This image is organized as a rectangular grid of dots called picture elements, or pixels. Each Pixel has an assigned value or color. The number of colors a pixel may have depends on its size or chunkiness. Two sizes are possible: four-color and sixteen-color. Exactly which colors map into the various pixel values is determined by a color table. This will be described later.

Pixel Size in the display is controlled independently for each scan line. Each scan line has a scan line control byte (SCB) which determines the scan line's properties. See Appendix B for more details.

Pixels are frequently thought of as points in the Cartesian coordinate system, with each pixel assigned a horizontal and vertical coordinate. Following the QuickDraw standard, the

coordinate grid falls between, rather than on pixels (see Figure 1). Each pixel is associated with the point that is above and to the left of it.

Figure 1
Pixels, Points and Rectangles



This scheme allows a rectangle to divide pixels into two classes: those which fall within the rectangle and those which fall outside the rectangle. Calls which draw rectangles only affect the pixels which fall inside the rectangle.

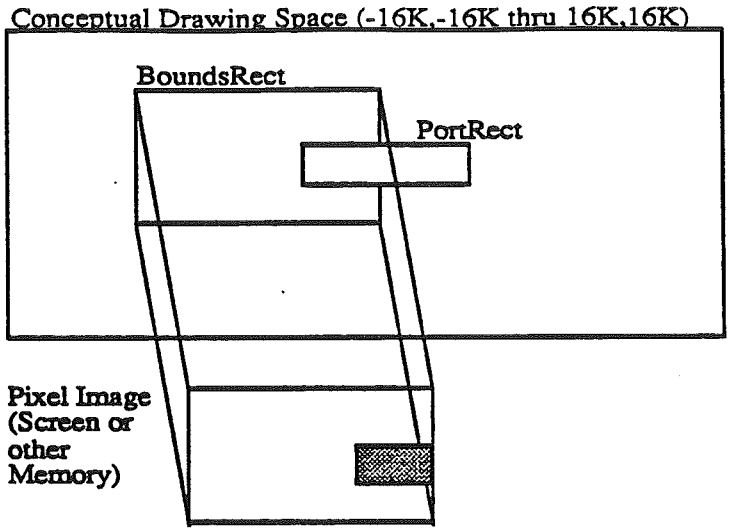
A pixel map need not be the area of memory associated with the graphics screen. QuickDraw II can treat other memory as pixel map memory and draw into it as easily as the screen memory.

Drawing can be done in coordinates appropriate to the data being used. Data is mapped from drawing space to the pixel map according to the information kept in two rectangles (see Figure 2):

a. The Bounds Rectangle
b. The Port Rectangle

An important point to note is that the conceptual drawing space for QuickDraw II is not the same as for QuickDraw on the Macintosh. On the Macintosh the drawing space is 64K by 64K pixels centered around 0,0. The bounds of the drawing space is -32K,-32k and 32K,32K. In QuickDraw II the drawing space is only 32K by 32K pixels. The bounds of drawing are -16K,-16K and 16K,16K. Commands to draw oudside this space will produce unpredictable results. They will not generate errors.

Figure 2. The Bounds and Port Rectangles



Active Port Rect (intersection of the BoundsRect and PortRect)

The BoundsRect is a rectangle that encompasses the entire pixel map. The upper left hand corner of the Bounds rect is the point that is above and to the left of the first pixel in the pixel map.

The PortRect is a rectangle that describes the "active" region of the pixel map. The intersection of the Port and Bounds rects is the only place that pixels in the pixel map will change (ignoring the VisRgn and ClipRgn to be discussed later).

A SetOrigin call allows you to change the position of both these rectangles in the coordinate plane. The rectangles remain the same size and in the same location relative to each other but the upper left hand corner (the origin) of the PortRect is set to the point passed by SetOrigin.

Drawing is the process by which pixels are altered in a pixel map. You may imagine a pen drawing the image by placing dots of the appropriate color at each pixel which falls under its path as specified by the program.

Drawings are clipped when instructions to draw in inactive parts of the drawing space are ignored. For example, if I am clipping to a rectangle defined by (100,100) and (200,200) and I try to draw a line from (0,0) to (1000,1000), only the pixels that fall inside the (100,100) thru (200,200) range are affected.

QuickDraw II also provides for clipping to arbitrary regions. Drawings are clipped to the intersection of two regions: the clip region (a user-maintained clipping region) and the vis region (a system-maintained clipping region). These work exactly as they do on the Macintosh.

The Drawing Environment

The drawing environment is a set of rules which explain how drawing actions behave. The environment includes information about where drawing will occur (what part of memory, its chunkiness), in what coordinate system, how it will be clipped, the pen state, the font state and some other stuff. The various parts of the drawing environment are described below.[2]

Drawing Location

QuickDraw II will draw anywhere in memory. The most common location may be the super hi-res screen, but a pixel map anywhere in memory and almost any size is acceptable as long as the entire destination pixel map is in a single bank.

PortSCB -- Flag to indicate chunkiness of pixel map and master color palette.

Pointer to the pixel map -- points to the first byte in the pixel map.

Width -- num bytes in a row of pixels (QuickDraw term is rowBytes).

Bounds Rectangle -- Rectangle that describes the extent of the pixel map and imposes a coordinate system on it.

Port Rectangle -- Rectangle that describes the active area of Data space.

Pen State

QuickDraw II maintains a graphics pen (position and size). Its position is used for drawing text, and its size is used for determining the size of a frame. There are two kinds of drawing: normal drawing and erasing. In normal drawing, the destination pixel map depends on what it was to start with, the original fill pattern or pixel image and the drawing mode.[3] Erasing just fills the affected pixels with the background pattern.

Pen Location -- A point in data space.

Pen Size -- A point describing the width and height of the pen.

Pattern Transfer Mode -- One of the 8 transfer modes supported by the Primitives. This mode is used when drawing horizontal lines with the fill pattern.

Pen Pattern -- The pen pattern is used when drawing graphic objects.

Drawing Mask -- The drawing mask prevents pixels aligned with zeros in the mask from begin altered during drawing operations which use patterns.

---

[2]The specific state of the display hardware (current color tables and SCBs) could also be though of as part of the environment, but for the purposes of this discussion we do not consider these.

[3]There are eight different drawing modes. These modes are rules used to derive the color of a pixel that is being drawn into. The eight modes and how they work are described in detail Appendix A.

Background Pattern -- The background pattern is used when erasing graphic objects.

Clipping

As stated earlier, drawing is clipped to a variety of rectangles and regions.

Other Stuff

QuickDraw II's local environment includes clipping information, handles to pictures, regions and polygons, as well as a pointer to GrafProcs record. The GrafProcs record is a record that holds pointers to all the standard drawing functions. A programmer may change the pointers in this record and cause different drawing routines to be used.

An entire drawing environment is kept in a single record (called the GrafPort) which can be saved and restored with a single call. This allows for simple (and hopefully fast) context switching. The GrafPort is a private data structure. The programmer can only change it by making calls to QuickDraw procedures which affect it.

Note:

This is different from QuickDraw on the Macintosh today where you can change fields in the GrafPort directly. (Macintosh QuickDraw will most likely evolve to this in the future.) The full GrafPort definition is provided for debugging purposes but programs should not rely on fields staying in the same place within the record.

Data Structures

Fixed Point or Fixed

F	long integer where high word represents a integer value and low word is a fractional value. Fractional part is always positive (for example $8000 is one half).

Pointer

P	long integer (highest byte must be zero)

Handle

H	long integer (highest byte must be zero)

Point

V	integer
H	integer

Rect

V1	integer
H1	integer
V2	integer
H2	integer

String

Standard ProDOS string starting with a length byte followed by up to 255 characters of data.

CString

ASCII characters terminated with a zero byte.

An_SCB_Byte

Bits	Meaning
0-3	Color Table
4	Reserved
5	Fill 0=off 1=on
6	Interrupt 0 = off 1 = on
7	Color Mode 0=320 1=640

ColorValue

```
Blue : 0..15
Green : 0..15
Red : 0..15
Reserved 0..15
```

Total size is one word. The word is arranged as follows:

```
| x | R | G | B |
 ------- -------
   high    low
   byte    byte
```

ColorTable

```
packed array [0..15] of ColorValue
```

LocInfo

```
PortSCB : an_scb_byte
reserved : byte
PointerToPixelImage : pointer
Width : integer
BoundsRect : rect
```

The width represents the number of bytes in a row (slice) of the pixel map. This number must be a multiple of 8.

```
nibble = 0..15
twobit = 0..3
bit = 0..1
```

Pattern

```
case mode of
        mode320:
                ( packed array [0..63] of nibble);
        mode640:
                ( packed array [0..63] of twobit);
```

Mask

packed array [0..63] of bit;

TextStyle

word where the following bits are defined:

0	bold
1	italic
2	underline

Bold and underline are supported in ROM.

PenState

```
PnLoc : point
PnSize : point
PnMode : integer
PnPat : pattern
PnMask : mask
```

GrafPort (Total Size is $AA or 170 bytes)

PortInfo : LocInfo
PortRect : rect
ClipRgn : handle
VisRgn : handle
BkPat : Pattern
PnLoc : Point
PnSize : Point
PnMode : integer
PnPat : pattern
PnMask : mask
PnVis : integer
FontHandle : Handle
FontID : Long
FontFlags : integer
TxSize : integer
TxFace : Style
TxMode : integer
SpExtra : fixed
ChExtra : fixed
FGColor : integer
BGColor : integer

PicSave : handle
RgnSave : handle
PolySave : handle
GrafProcs : pointer

ArcRot : integer

UserField : long
SysField : long

Font

See Appendix D

FontInfoRecord

Ascent	: integer
Descent	: integer
WidMax	: integer
Leading	: integer

FontGlobalsRecord (currently 12 bytes long)

fontID	: integer
style	: textstyle
size	: integer
version	: VersionNumber
widMax	: integer
fbrExtent	: integer

BufSizeRecord (8 bytes long)

```
MaxWidth        : integer
TextBufHeight   : integer
TextBufRowWords : integer
FontWidth       : integer
```

Polygon

```
PolySize        : integer
PolyBBox        : rect
PolyPoints      : array [0..?] of point
```

Cursor

```
CursorHeight    : integer
CursorWidth*    : integer
CursorImage     : [array 1..CursorHeight,1..CursorWidth] of word
CursorMask      : [array 1..CursorHeight,1..CursorWidth] of word
HotSpotY        : integer
HotSpotX        : integer
```

*This is number of words wide of a single horizontal slice of the cursor. The last word in each slice of the cursor must be 0.

Arror Cursor in 320 Mode:

```
        dc i'11,4'                          ; eleven slices
*                                           ; by 4 words

        dc h'0000000000000000'              ; cursor image
        dc h'0f00000000000000'
        dc h'0FF0000000000000'
        dc h'0FFF000000000000'
        dc h'0FFFF00000000000'
        dc h'0FFFFF0000000000'
        dc h'0FFFFFF000000000'
        dc h'0FFFFFFF00000000'
        dc h'0FF0FF0000000000'
        dc h'00000FF000000000'
        dc h'0000000000000000'

        dc h'FF00000000000000'              ; mask image
        dc h'FfF0000000000000'
        dc h'FFFF000000000000'
        dc h'FFFFF00000000000'
        dc h'FFFFFF0000000000'
        dc h'FFFFFFF000000000'
        dc h'FFFFFFFF00000000'
        dc h'FFFFFFFFF0000000'
        dc h'FFFFFFFF00000000'
        dc h'FFF0FFFF00000000'
        dc h'00000FFF00000000'

        dc i'1,1'                           ; hot spot
```

The Calls

Each of the calls listed below is listed in the form:

ToolCall Brief description of the purpose of the function.

```
Stack Before Call
      |   previous contents   |
      |   Space for Result    |   Space for result (if any).
      |       Param1          |   Description of parameter
      |       Param2          |   Description of parameter
      |                       |<-SP

Stack After Call
      |   previous contents   |
      |     The Result        |   Result (if any).
      |                       |<-SP
```

Further description of the function and the parameters, if necessary.

A call is made as follows:

1. If the function has any output, push room for it on the stack.
2. Push the inputs in the order listed.
3. Invoke the macro for the call you want to make. The macro loads x with the appropriate value and executes a JSL to the tool dispatcher.
4. If the function returned any output it is now at the top of the stack.

Warning

QuickDraw II is very unforgiving of certain kinds of programming errors.

1) Making any QuickDraw calls without initializing QuickDraw First.
2) Passing any bad handles to QuickDraw
3) Making any QuickDraw calls with a bad port.

Error codes are not likely to be returned when these three kinds of errors are made. In fact there is no guarentee that your program will get control ever again after making one of these errors.

Unfortunately, a worse situation is that your program will appear to work for a while even though you make one of these errors but later die horribly for no reason.

Housekeeping Functions

QDBootInit Initializes QuickDraw II at boot time. The function puts the address of the cursor update routine into the bank E1 vectors. An application should never make this call.

No Inputs.

QDStartup Initializes Quickdraw II, sets the current port to the standard port, and clears the screen.

Stack Before Call

previous contents	
ZeroPageLoc	integer
MasterSCB	SCB (word)
MaxWidth	integer
ProgramID	User ID for memory manager (word).
	<-SP

Stack After Call

previous contents	
	<-SP

QuickDraw uses three consecutive pages of bank zero for its zero page starting at the specified address. The *MasterSCB* is used to set all SCB's in the super hi-res graphics screen. *MaxWidth* is a number that tells QuickDraw II the size in bytes of the largest pixel map that will be drawn to (a value of zero indicates screen width). This allows QuickDraw II to allocate certain buffers it needs only once and keep them throughout the life of the application. *ProgramID* is the ID QuickDraw II will use when getting memory from the Memory Manager. All memory is reserved in the name of this ID.

This call can fail for several reasons. The most common reasons are QuickDraw is already initialized, and there is not enough memory available for QuickDraw to obtain the buffers it needs.

Warning:

QuickDraw now uses three pages in bank zero.

Error Codes

AlreadyInitialized	This is returned when an attempt is made to initialize QuickDraw a second time without shutting down first.
ScreenReserved	This is returned when an the memory manager reports that the screen memory (bank E1 from $2000 to $9FFF) is already owned by someone else.
Memory Manager Errors	Any errors from the memory manager are returned unchanged.

QDShutDown Frees up any buffers that were allocated. This call returns an error if QuickDraw is not active when the call was made.

No Inputs.

Possible Errors

NotActive	This is returned when an attempt is made to shutdown QuickDraw without ever starting it up.
Memory Manager Errors	Any errors from the memory manager are returned unchanged.

QDVersion

Returns the version of QuickDraw II.

Stack Before Call

previous contents	
Space for Version	**word**
	<-SP

Stack After Call

previous contents	
Version	**word**
	<-SP

Possible Errors

None.

QDStatus

Returns whether or not QuickDraw is active.

Stack Before Call

previous contents	
Space for Status	**Boolean (word)**
	<-SP

Stack After Call

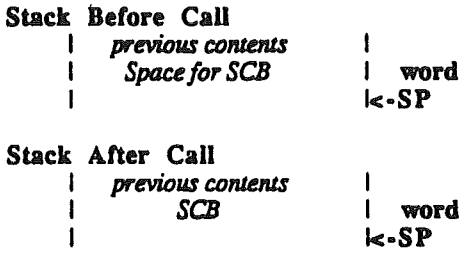
previous contents	
Status	**Boolean (word)**
	<-SP

Possible Errors

None.

Global Environment Calls

GetStandardSCB Returns a copy of the standard SCB in the low byte of the word.



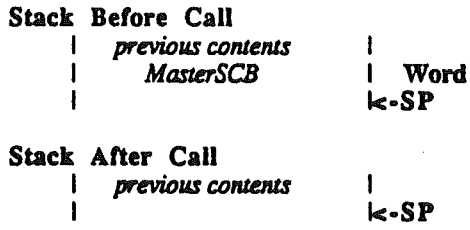
The SCB has the following fields:

Bits	Meaning
0-3	Color Table 0
4	Reserved
5	Fill off
6	Interrupt off
7	Color Mode = 320

Possible Errors

None.

SetMasterSCB Sets the master SCB to the specified value (only the low byte is used).



The master SCB is the global mode byte used throughout QuickDraw II. The master SCB is used by routines like InitPort to decide what standard values should be put into the GrafPort.

Possible Errors

None

GetMasterSCB Returns a copy of the master SCB (only the low byte is valid).

```
Stack Before Call
     |   previous contents     |
     |   Space MasterSCB       |   word
     |                         |<-SP

Stack After Call
     |   previous contents     |
     |     MasterSCB           |   word
     |                         |<-SP
```

Possible Errors

None

InitColorTable Returns a copy of the standard color table for the current mode.

```
Stack Before Call
     |   previous contents     |
     |     TablePointer        |   Pointer to color table.
     |                         |<-SP

Stack After Call
     |   previous contents     |
     |                         |<-SP
```

The entries are as follows for 320 mode:

Pixel Value	Name	Master Color	
0	Black	0 0 0	Opposite of White
1	Dark Gray	7 7 7	
2	Brown	8 4 1	
3	Purple	7 2 C	
4	Blue	0 0 F	
5	Dark Green	0 8 0	
6	Orange	f 7 0	
7	Red	D 0 0	
8	Flesh	F A 9	
9	Yellow	F F 0	
10	Green	0 E 0	
11	Light Blue	4 D F	
12	Lilac	D A F	
13	Periwinkle Blue	7 8 F	
14	Light Gray	C C C	
15	White	F F F	Opposite of Black

The entries are as follows for 640 mode:

Pixel Value	Name	Master Color
0	Black	0 0 0
1	Red	F 0 0
2	Green	0 F 0
3	White	F F F
4	Black	0 0 0
5	Blue	0 0 F
6	Yellow	F F 0
7	White	F F F
8	Black	0 0 0
9	Red	F 0 0
A	Green	0 F 0
B	White	F F F
C	Black	0 0 0
D	Blue	0 0 F
E	Yellow	F F 0
F	White	F F F

For the rationale behind this color palette see Art Cabral's paper on Dithering Techniques in 640 mode.

Possible Errors

None.

SetColorTable Sets a color table to specified values.

Stack Before Call

```
|   previous contents   |
|     TableNumber       |  integer
|       DestPtPtr       |  Pointer to color table
|                       |<-SP
```

Stack After Call

```
|   previous contents   |
|                       |<-SP
```

TableNumber identifies the table to be set to the values specified in the table pointed to. The 16 color tables are stored starting at $9E00. Each table takes $20 bytes. Each word in the table represents one of 4096 colors. The high nibble of the high byte is ignored.

Possible Errors

BadTable	Only Table numbers from 0 to 15 are valid.

GetColorTable Fills a color table with the contents of another color table.

Stack Before Call

```
|   previous contents   |
|     TableNumber       |  integer
|       DestPtPtr       |  Pointer to color table
|                       |<-SP
```

Stack After Call

```
|   previous contents   |
|                       |<-SP
```

TableNumber specifies the number of the color table whose contents are to be copied; *TablePtr* points to the color table which is to receive the contents.

Possible Errors

BadTable	Only Table numbers from 0 to 15 are valid.

SetColorEntry Sets the value of a color in a specified color table.

Stack Before Call

```
|   previous contents   |
|     TableNumber       |  integer
|     EntryNumber       |  integer
|       NewColor        |  integer
|                       |<-SP
```

Stack After Call

```
|   previous contents   |
|                       |<-SP
```

TableNumber specifies the number of the color table; *EntryNumber* specifies the number of the color to be changed; *Value* sets the color.

Possible Errors

BadTable	Only Table numbers from 0 to 15 are valid.

GetColorEntry Returns the value of a color in a specified color table .

```
Stack Before Call
     |  previous contents  |
     |   Space for Color   |  word
     |    Table Number     |  integer
     |    Entry Number     |  integer
     |                     |<-SP

Stack After Call
     |  previous contents  |
     |        Color        |  word
     |                     |<-SP
```

TableNumber specifies the number of the color table; *EntryNumber* specifies the number of the color to be examined; *Value* returns the color.

Possible Errors

BadTable	Only Table numbers from 0 to 15 are valid.

SetSCB Sets the scan line control byte (SCB) to a specified value.

```
Stack Before Call
     |  previous contents  |
     |       ScanLine     |  integer
     |       NewSCB       |  integer
     |                     |<-SP

Stack After Call
     |  previous contents  |
     |                     |<-SP
```

Scanline identifies the scan line whose SCB is to be set; *Value* sets the SCB.

Possible Errors

BadScanLine	Only scan line numbers from 0 to 199 are valid.

GetSCB Returns the value of a specified scan line control byte (SCB).

```
Stack Before Call
     |  previous contents  |
     |   Space for SCB     |  word
```

```
|      ScanLine        |  integer
|                      |<-SP

Stack After Call
|  previous contents   |
|       SCB            |  word
|                      |<-SP
```

Scanline identifies the scan line whose SCB is to be examined; *Value* returns the value of the SCB.

Possible Errors

BadScanLine	Only scan line numbers from 0 to 199 are valid.

SetAllSCBs Sets all scan line control bytes (SCBs) to a specified value.

```
Stack Before Call
|  previous contents   |
|      NewSCB          |  word
|                      |<-SP

Stack After Call
|  previous contents   |
|                      |<-SP
```

Possible Errors

None

SetSysFont Tells QuickDraw to use the font passed as a system font.

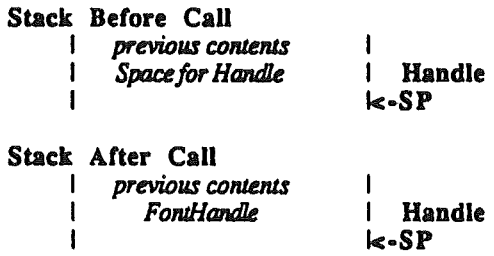
```
Stack Before Call
|  previous contents   |
|     FontHandle       |  Handle to font that will be system font.
|                      |<-SP

Stack After Call
|  previous contents   |
|                      |<-SP
```

There is a default system font in ROM which will be used unless this call is made. A handle to the system font is put in the every port opened or inited.

Possible Errors

None

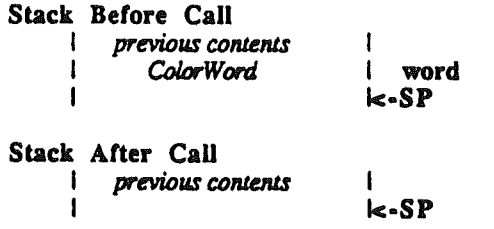
GetSysFont Returns a handle to the current system font.



Possible Errors

None

ClearScreen Sets the words in the screen memory to the value passed.



This is a very fast clear screen that just stuffs the value passed into each word of screen memory. The color you see on the screen will not be a solid color unless all the pixels in the word passed are the same.

Possible Errors

None

GrafOn Turns on the super hi-res graphics mode.

This routine only touches the bit in the NewVideo softswitch that affects the what is displayed. It does not change the linearization bit in the field.

No Inputs.

Possible Errors

None

GrafOff Turns off the super hi-res graphics mode.

This routine only touches the bit in the NewVideo softswitch that affects the what is displayed. It does not change the linearization bit in the field.

No Inputs.

Possible Errors

None

GrafPort Calls

OpenPort Initializes specified memory locations as a standard port and allocates new VisRgn and ClipRgn.

```
Stack Before Call
    |  previous contents     |
    |       PortPtr          |  Pointer to Port.
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |                        |<-SP
```

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

InitPort Initializes specified memory locations as a standard port.

```
Stack Before Call
    |  previous contents     |
    |       PortPtr          |  Pointer to Port.
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |                        |<-SP
```

InitPort, unlike **OpenPort**, assumes that the region handles are valid and does not allocate new handles. Otherwise, **InitPort** performs the same functions.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

ClosePort Deallocates the memory associated with a port.

```
Stack Before Call
    |  previous contents     |
    |       PortPtr          |  Pointer to port.
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |                        |<-SP
```

All handles are discarded. If the application disposes of the memory containing the port without first calling **ClosePort**, the memory associated with the handles is lost and cannot be claimed.

Warning: Never close the current port.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

SetPort Makes the specified port the current port.

```
Stack Before Call
     |  previous contents     |
     |     .PortPtr           |  Pointer to port.
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

Possible Errors

None

GetPort Returns the handle to the current port.

```
Stack Before Call
     |  previous contents     |
     |  Space for PortPtr     |  Space for Pointer to port.
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |      PortPtr           |  Pointer to port.
     |                        |<-SP
```

Possible Errors

None

SetPortLoc Sets the current port's map information structure to the specified location information.

```
Stack Before Call
     |  previous contents     |
     |   Ptr to LocInfo       |  Pointer
     |                        |<-SP

Stack After Call
```

```
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

GetPortLoc Gets the current port's map information structure and puts it at the address indicated.

```
Stack Before Call
|  previous contents  |
|   Ptr to LocInfo    |  Pointer
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

SetPortRect Sets the current port's port rectangle to the specified recangle.

```
Stack Before Call
|  previous contents  |
|      RectPtr        |  Pointer to rectangle.
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

GetPortRect Returns the current port's map port rectangle.

```
Stack Before Call
|  previous contents  |
|      RectPtr        |  Pointer to rectangle.
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

SetPortSize Changes the size of the current GrafPort's PortRect.

```
Stack Before Call
     |  previous contents     |
     |        width           |  integer
     |        height          |  integer
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

This does not affect the pixel map, but just changes the active area of the GrafPort. The call is normally used by the Window Manager.

Possible Errors

None

MovePortTo Changes the location of the current GrafPort's PortRect.

```
Stack Before Call
     |  previous contents     |
     |        H               |  integer
     |        V               |  integer
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

This does not affect the pixel map, but just changes the active area of the GrafPort. The call is normally used by the Window Manager.

Possible Errors

None

SetOrigin Adjusts the contents of PortRect and BoundsRect so that the upper left corner of PortRect is set to the specified point.

```
Stack Before Call
     |  previous contents     |
     |        H               |  integer
     |        V               |  integer
     |                        |<-SP
```

```
Stack After Call
    |  previous contents     |
    |                        |<-SP
```

VisRgn is also affected, but ClipRgn is not. The pen position does not change.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged. (The memory manager may be called when the VisRgn is offset).

SetClip Sets the clip region to the region passed by using CopyRgn.

```
Stack Before Call
    |  previous contents     |
    |     RgnHandle          |  Handle
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |                        |<-SP
```

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

GetClip Copies the Clip Region to the region passed. The region must have been created earlier with a new rgn call.

```
Stack Before Call
    |  previous contents     |
    |     RgnHandle          |  Handle
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |                        |<-SP
```

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

ClipRect Changes the clip region of the current GrafPort to a rectangle equivalent to a given rectangle.

```
Stack Before Call
    |  previous contents     |
    |       RectPtr          |  Pointer
```

```
|                          |<-SP

Stack After Call
|  previous contents       |
|                          |<-SP
```

This does not change the region handle, but affects the region itself.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

HidePen Decrements the pen level. A positive pen level indicates drawing will occur; a negative pen level indicates drawing will not occur.

No Inputs.

Possible Errors

None

ShowPen Increments the pen level. A positive pen level indicates that drawing will occur; a negative pen level indicates drawing will not occur.

No Inputs.

Possible Errors

None

GetPen Returns the pen location.

```
Stack Before Call
|  previous contents       |
|     PointPtr             |  Pointer to point.
|                          |<-SP

Stack After Call
|  previous contents       |
|                          |<-SP
```

Possible Errors

None

SetPenState Sets the pen state in the GrafPort to the values passed.

Stack Before Call

```
|  previous contents  |
|    PenStatePtr      |  Pointer to PenState record.
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

GetPenState Returns the pen state from the GrafPort.

Stack Before Call

```
|  previous contents  |
|    PenStatePtr      |  Pointer to PenState record.
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

SetPenSize Sets the current pen size to the specified pen size.

Stack Before Call

```
|  previous contents  |
|       width         |  integer
|       height        |  integer
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

GetPenSize Returns the current pen size at the place indicated.

Stack Before Call

```
|  previous contents  |
|      PointPtr       |  Pointer to point.
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
```

```
|                          |<-SP
```

Possible Errors

None

SetPenMode Sets the current pen mode to the specified pen mode.

```
Stack Before Call
|  previous contents  |
|     PenMode         |  word
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

GetPenMode Returns the pen mode from the current port.

```
Stack Before Call
|  previous contents  |
|  Space for PenMode  |  Space for PenMode (word).
|                     |<-SP

Stack After Call
|  previous contents  |
|     PenMode         |  word
|                     |<-SP
```

Possible Errors

None

SetPenPat Sets the current pen pattern to the specified pen pattern.

```
Stack Before Call
|  previous contents  |
|     PatternPtr      |  Pointer to pattern.
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

GetPenPat Returns the current pen pattern at the specified location.

```
Stack Before Call
     |  previous contents     |
     |     PatternPtr         |  Pointer to pattern.
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

Possible Errors

None

SetSolidPenPat Sets the pen pattern to a solid pattern using the specified color.

```
Stack Before Call
     |  previous contents     |
     |     ColorNum           |  integer
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

Only an appropriate number of bits in *ColorNum* are used. If the PortSCB indicates 320 mode, then four bits are used; if the PortSCB indicates 640 mode, then two bits are used.

Possible Errors

None

SetPenMask Sets the pen mask to the specified mask.

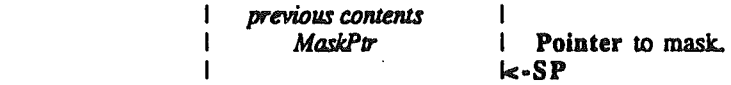
```
Stack Before Call
     |  previous contents     |
     |     MaskPtr            |  Pointer to mask.
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

Possible Errors

None

GetPenMask Returns the pen mask at the specified location.

Stack Before Call



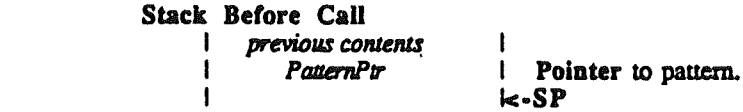
Stack After Call

Possible Errors

None

SetBackPat Sets the background pattern to the specified pattern.

Stack Before Call



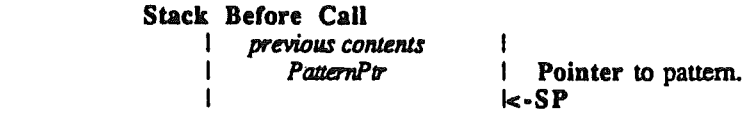
Stack After Call

Possible Errors

None

GetBackPat Returns the background pattern at the specified location.

Stack Before Call



Stack After Call

Possible Errors

None

SetSolidBackPat Sets the background pattern to a solid pattern using the specified color.

```
Stack Before Call
    |  previous contents    |
    |     ColorNum          |  integer
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Only an appropriate number of bits in *ColorNum* are used. If the PortSCB indicates 320 mode, then four bits are used; if the PortSCB indicates 640 mode, then two bits are used.

Possible Errors

None

SolidPattern Sets the spcified pattern to a solid pattern using the specified color.

```
Stack Before Call
    |  previous contents    |
    |     PatternPtr        |  Pointer
    |     ColorNum          |  integer
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Only an appropriate number of bits in *ColorNum* are used. If the PortSCB indicates 320 mode, then four bits are used; if the PortSCB indicates 640 mode, then two bits are used.

Possible Errors

None

PenNormal Sets the pen state to the standard state (PenSize = 1,1; PenMode = copy; PenPat = Black; PenMask = 1's). The pen location is not changed.

No Inputs.

Possible Errors

None

MoveTo Moves the current pen location to the specified point.

```
Stack Before Call
    |  previous contents  |
    |        H            |  integer
    |        V            |  integer
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

Possible Errors

None

Move Moves the current pen location by the specified horizontal and vertical displacements.

```
Stack Before Call
    |  previous contents  |
    |        dH           |  integer
    |        dV           |  integer
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

Possible Errors

None

SetFont Sets the current font to the specified font.

```
Stack Before Call
    |  previous contents  |
    |   NewFontHandle     |  Handle to Font.
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

Possible Errors

None

GetFont Returns a handle to the current font.

```
Stack Before Call
    |  previous contents    |
    |  Space for Result     |  Space for Handle.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |    FontHandle         |  Handle
    |                       |<-SP
```

Possible Errors

None

SetFontID Sets the fontID field in the GrafPort.

```
Stack Before Call
    |  previous contents    |
    |    NewFontID          |  FontID (long).
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

This routine does not change the current font. All it does is set the font ID. This call is designed for use by the font manager for the benefit of the picture routines. The picture routines use this field to try to find the font the application really wanted to draw with, rather than the one that was available when the picture was recorded.

Possible Errors

None

GetFontID Returns the FontID field in the GrafPort.

```
Stack Before Call
    |  previous contents    |
    |  Space for Result     |  Space for FontID (long).
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |    FontHandle         |  FontID (long)
    |                       |<-SP
```

Possible Errors

None

GetFontInfo Returns information about the current font in the specified record.

```
Stack Before Call
    |  previous contents    |
    |      FIRecPtr         |  Pointer FontInfo record.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

The information in the FontInfoRecord does not reflect current style modifications nor chExtra and spExtra.

Possible Errors

None

GetFGSize Returns the size of the font globals record.

```
Stack Before Call
    |  previous contents    |
    |  Space for integer    |  Space for resulting size
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |         size          |  integer
    |                       |<-SP
```

We expect that the amount of information returned by the GetFontGlobals call will change from version to version. This call makes it possible to find out how many bytes are being returned.

For the version of QuickDraw in ROM, this value is 10. For the RAM patch (version 1.1) the value is 12.

Possible Errors

None

GetFontGlobals Returns information about the current font in the specified record.

```
Stack Before Call
    |  previous contents    |
    |      FGRecPtr         |  Pointer FontGlobals record.
```

```
|                          |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

The size of the FontGlobals record is returned by the GetFGSize call. The infomation represents the grafport's current font, and does not reflect style modifications, chExtra or spExtr etc. Future versions of QuickDraw II may add more information at the end of this record (see GetFGSize), but the current fields and their order will be maintained.

Possible Errors

None

SetFontFlags Sets the font flags to the specified value.

```
Stack Before Call
    |  previous contents    |
    |    FontFlags          |  word
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

The font flags are used to indicate special operations performed on the text. Only one is defined at this time. If the lowest bit of the word is set then the font will be considered a fixed with font (rather than proportional) and all characters will be equally spaced. The width of the characters is that of widMax. Other flags are reserved for future use.

Possible Errors

None

GetFontFlags Returns the current font flags.

```
Stack Before Call
    |  previous contents    |
    |  Space for Flags      |  Space for word.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |    FontFlags          |  word
    |                       |<-SP
```

Possible Errors

None

SetTextFace Sets the text face to the specified value.

```
Stack Before Call
    |  previous contents     |
    |      TextFace          |  word
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |                        |<-SP
```

Up to sixteen operations are possible on the text (only two are supported in ROM). Each bit in the textface word represents a different face. Plain text has a text face of 0.

Plain	TextFace = 0
Bold	TextFace = 1
Underline	TextFace = 2

Possible Errors

None

GetTextFace Returns the current text face.

```
Stack Before Call
    |  previous contents     |
    |  Space for Result      |  Space for word.
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |      TextFace          |  word
    |                        |<-SP
```

Possible Errors

None

SetTextMode Sets the text mode to the specified value.

```
Stack Before Call
    |  previous contents     |
    |      TextMode          |  word
    |                        |<-SP

Stack After Call
    |  previous contents     |
```

```
|                          |<-SP
```

There are eight different text transfer modes (and their opposites). The fastest modes are the modes which only transfer the foreground to the destination. The fastest of the foreground modes are FG-OR and FG-XOR. FG-BIC is almost as fast and FG-COPY is the slowest.

Possible Errors

None

GetTextMode Returns the current text mode.

```
Stack Before Call
    |  previous contents     |
    |   Space for Result     |  Space for word.
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |     TextMode           |  word
    |                        |<-SP
```

Possible Errors

None

SetSpaceExtra Sets the space extra field in the grafport to the specified value.

```
Stack Before Call
    |  previous contents     |
    |     SpaceExtra         |  Fixed Point Value
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |                        |<-SP
```

The space extra is used by programs that are trying to justify text to a left and right boundry. When the space extra field is non zero, its value is added the width of each space printed in a string. It is a fixed point value rather than an integer because you want to be able to control this down to the last pixel.

Possible Errors

None

GetSpaceExtra Returns the space extra field from the grafport.

```
Stack Before Call
    |  previous contents     |
```

```
|  Space for Result     |  Space for Fixed Point Value.
|                       |<-SP
```

Stack After Call

```
|  previous contents    |
|    SpaceExtra         |  LONG
|                       |<-SP
```

Possible Errors

None

SetCharExtra Sets the char extra field in the grafport to the specified value.

Stack Before Call

```
|  previous contents    |
|    CharExtra          |  Fixed Point Value
|                       |<-SP
```

Stack After Call

```
|  previous contents    |
|                       |<-SP
```

The CharExtra field in the grafport is used to add width to every character in the font that has width. It does not affect zero width characters. This field is present because some fonts that look good in one graphics mode need a little extra space between the characters in another mode.

Possible Errors

None

GetSpaceExtra Returns the space extra field from the grafport.

Stack Before Call

```
|  previous contents    |
|  Space for Result     |  Space for Fixed Point Value.
|                       |<-SP
```

Stack After Call

```
|  previous contents    |
|    CharExtra          |  LONG
|                       |<-SP
```

Possible Errors

None

SetForeColor Sets the foreground color field in the grafport to the specified value.

Stack Before Call

```
|  previous contents  |
|    ForeColor        |  word
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

GetForeColor Returns the current foreground color from the grafport.

Stack Before Call

```
|  previous contents  |
|  Space for Result   |  Space for ForeColor (word).
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
|    ForeColor        |  word
|                     |<-SP
```

Possible Errors

None

SetBackColor Sets the background color field in the grafport to the specified value.

Stack Before Call

```
|  previous contents  |
|    BackColor        |  word
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

GetBackColor Returns background color field from the grafport.

Stack Before Call

```
|  previous contents  |
|  Space for Result   |  Space for BackColor (word).
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
|    BackColor        |  word
```

```
|                              |<-SP
```

Possible Errors

None

SetBufDims This sets the size of the QuickDraw II clipping and text buffers. It overrides the MaxWidth value supplied to QDStartup and the text buffer defaults set at that time.

```
Stack Before Call
    |  previous contents    |
    |     MaxWidth          |  integer
    |   MaxFontHeight       |  integer
    |   MaxFBRExtent        |  integer
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

MaxWidth is the width, in bytes, of the widest pixelmap the application will draw into (0 indicates screen width). MaxFontHeight is the height of the tallest font the application will use (fRectHeight from the font record, computable as ascent + descent from the GetFontInfo call). MaxFBRExtent is the greatest fbrExtent value of any font the application will use. fbrExtent is a field in the font record, and is returned by GetFontGlobals. It is defined as the greatest (horizontal) distance, in pixels, from the character origin to the furthest foreground or background pixel of any character in the font. For more information, see "Fonts And Text In QuickDraw II".

SetBufSize "pads" the text buffer to permit values of chExtra ≤ fbrExtent (of the currently active font), spExtra ≤ fbrExtent, and style modifications which add upto 36 pixels to the bounds width (width of foreground and background) of any character.

When QDStartUp is called, it makes an internal call to SetBufDims, with the following values:

MaxWidth as supplied by the application;
MaxFontHeight = 2 * (height of system font);
MaxFBRExtent = 2 * (fbrExtent of system font).

If the application does not use fonts larger than that, and only uses reasonable values of chExtra and spExtra and no very extreme (presumably customized) style modifications, then there is no reason to call SetBufDims or ForceBufDims.

Error Codes

Memory Manager Errors	Any errors from the memory manager are returned unchanged.

ForceBufDims This call performs the same function as SetBufDims, except that MaxFBRExtent value is not padded in any way. Any style modification, etc., that might increase the bounds width of a character must be added to MaxFBRExtent by the application.

```
Stack Before Call
     |  previous contents    |
     |     MaxWidth          |  integer
     |   MaxFontHeight       |  integer
     |   MaxFBRExtent        |  integer
     |                       |<-SP

Stack After Call
     |  previous contents    |
     |                       |<-SP
```

This can be used to allow absurdly large values of chExtra, spExtra, strange styles, and the like.

It is usually better to call SetBufDims and ForceBufDims once (if at all), early in the application with reasonable "max" values, because claiming and clearing a buffer can take lots of time. However, they may be called at any time. Also, it may sometimes be worthwhile to make one of these calls to shrink the size of the text buffer, thus conserving memory and (slightly) speeding performance. Don't be stingy, though.

Error Codes

Memory Manager Errors	Any errors from the memory manager are returned unchanged.

SaveBufDims This call saves QuickDraw's buffer sizing information in an 8 byte record supplied by the caller.

```
Stack Before Call
     |  previous contents    |
     |  Pointer to SizeInfo  |  Pointer
     |                       |<-SP

Stack After Call
     |  previous contents    |
     |                       |<-SP
```

This call can be used when an application (or tool) wants to temporarily change the size of the QuickDraw buffers but wants to be able to restore them as well.

Possible Errors

None

RestoreBufDims This call restores QuickDraw's internal buffers to the sizes described in an 8 byte record. This record is created by SaveBufDims.

```
Stack Before Call
    |  previous contents     |
    |    MaxWidth            |  integer
    |  MaxFontHeight         |  integer
    |  MaxFBRExtent          |  integer
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |                        |<-SP
```

This call can be used when an application (or tool) wants to temporarily change the size of the QuickDraw buffers but wants to be able to restore them as well.

Error Codes

Memory Manager Errors	Any errors from the memory manager are returned unchanged.

SetClipHandle Sets the clip region handle field in the grafport to the value passed.

```
Stack Before Call
    |  previous contents     |
    |    RgnHandle           |  Handle
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |                        |<-SP
```

Possible Errors

None

GetClipHandle Returns a copy of the handle to the ClipRgn.

```
Stack Before Call
    |  previous contents     |
    |  Space for Result      |  Space for Handle
    |                        |<-SP

Stack After Call
    |  previous contents     |
    |    RgnHandle           |  Handle
    |                        |<-SP
```

Possible Errors

None

SetVisRgn Sets the vis region to the region passed by using CopyRgn.

```
Stack Before Call
    |  previous contents    |
    |     RgnHandle         |  Handle
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

GetVisRgn Copies the contents of the VisRgn into the region passed. The region must have been created earlier with a NewRgn call.

```
Stack Before Call
    |  previous contents    |
    |     RgnHandle         |  Handle
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

SetVisHandle Sets the clip region handle field in the graf port to the value passed.

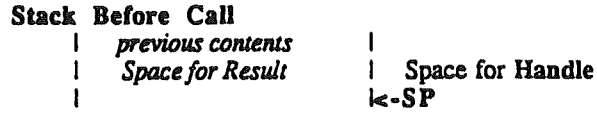
```
Stack Before Call
    |  previous contents    |
    |     RgnHandle         |  Handle
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

GetVisHandle Returns a copy of the handle to the VisRgn.

Stack Before Call



Stack After Call

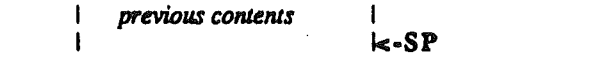
Possible Errors

None

SetPicSave Sets the picsave field to the value passed. This is an interal routine that should not be used by application programs.

Stack Before Call

Stack After Call



Possible Errors

None

GetPicSave Returns the contents of the PicSave field in the GrafPort.

Stack Before Call

Stack After Call

Possible Errors

None

SetRgnSave Sets the RgnSave field to the value passed. This is an interal routine that should not be used by application programs.

```
Stack Before Call
     |  previous contents    |
     |   RgnSaveValue        |  LONG
     |                       |<-SP

Stack After Call
     |  previous contents    |
     |                       |<-SP
```

Possible Errors

None

GetRgnSave Returns the contents of the RgnSave field in the GrafPort.

```
Stack Before Call
     |  previous contents    |
     |   Space for Long      |  Space for RgnSaveValue
     |                       |<-SP

Stack After Call
     |  previous contents    |
     |   RgnSaveValue        |  Long
     |                       |<-SP
```

Possible Errors

None

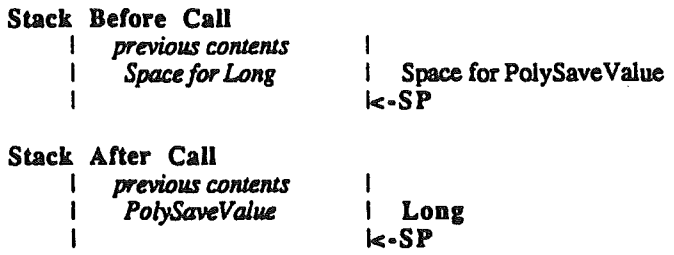
SetPolySave Sets the PolySave field to the value passed. This is an interal routine that should not be used by application programs.

```
Stack Before Call
     |  previous contents    |
     |   PolySaveValue       |  Long
     |                       |<-SP

Stack After Call
     |  previous contents    |
     |                       |<-SP
```

Possible Errors

None

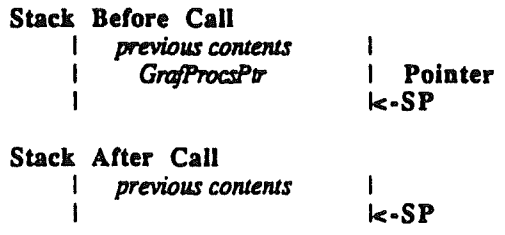
GetPolySave Returns the contents of the PicSave field in the GrafPort.



Possible Errors

None

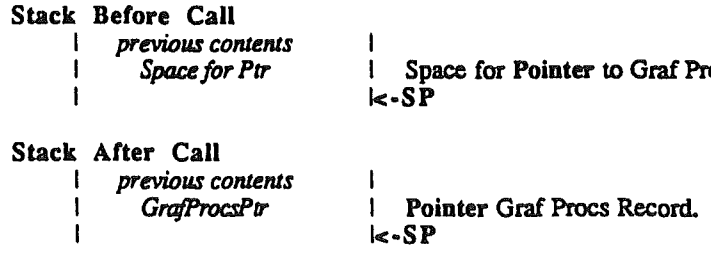
SetGrafProcs Sets the GrafProcs field to the value passed.



Possible Errors

None

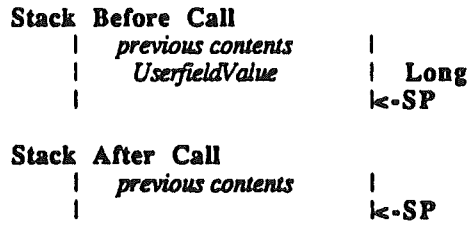
GetGrafProcs Returns the contents of the Pointer to the GrafProcs record associated with the GrafPort.



Possible Errors

None

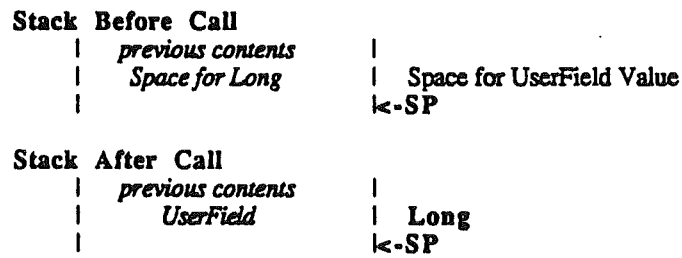
SetUserField Sets the UserField field in the GrafPort to the value passed. Programs can use this field to attach any data they want to a GrafPort by using this field as a pointer to some other data area.



Possible Errors

None

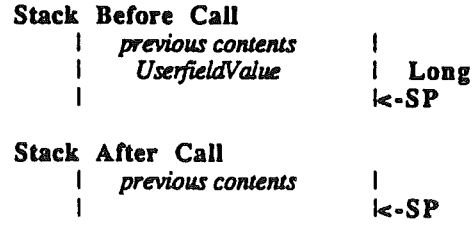
GetUserField Returns the contents of the UserField field in the GrafPort.



Possible Errors

None

SetSysField Sets the SysField field to the value passed. This is an interal routine that should not be used by application programs.

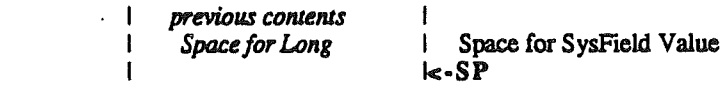


Possible Errors

None

GetSysField Returns the contents of the SysField field in the GrafPort.

Stack Before Call



Stack After Call

Possible Errors

None

Drawing Calls

LineTo Draws a line from the current pen location to the specified point.

```
Stack Before Call
     |  previous contents     |
     |        H               |  integer
     |        V               |  integer
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

Possible Errors

Memory Mgr Errors If a region is open, LineTo commands contribute to the definition. Memory manager errors can occur at this time.

Line Draws a line from the current pen location to a new point specified by the horizontal and vertical displacements.

```
Stack Before Call
     |  previous contents     |
     |        dH              |  integer
     |        dV              |  integer
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

Possible Errors

Memory Mgr Errors If a region is open, Line commands contribute to the definition. Memory manager errors can occur at this time.

Rectangle Drawing Calls

FrameRect Draws the boundary of the specified rectangle with the current pattern and pen size.

```
Stack Before Call
     |  previous contents     |
     |      RectPtr           |  Pointer to rectangle.
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

Only points entirely within the rectangle are affected.

Possible Errors

Memory Mgr Errors — If a region is open, FrameRect commands contribute to the definition. Memory manager errors can occur at this time.

PaintRect — Paints (fills) the interior of the specified rectangle with the current pen pattern.

```
Stack Before Call
     |  previous contents     |
     |       RectPtr          |  Pointer to rectangle.
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

Possible Errors

None

EraseRect — Erases the interior of the specified rectangle with the background pattern.

```
Stack Before Call
     |  previous contents     |
     |       RectPtr          |  Pointer to rectangle.
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

Possible Errors

None

InvertRect — Inverts the pixels in the interior of the specified rectangle.

```
Stack Before Call
     |  previous contents     |
     |       RectPtr          |  Pointer to rectangle.
     |                        |<-SP

Stack After Call
     |  previous contents     |
     |                        |<-SP
```

Possible Errors

None

FillRect Paints (fills) the interior of the specified rectangle with the specified pattern.

Stack Before Call

previous contents	
RectPtr	Pointer to rectangle.
PatternPtr	Pointer to pattern.
	<-SP

Stack After Call

previous contents	
	<-SP

Possible Errors

None

Drawing Regions

FrameRgn Draws the boundary of the specified region with the current pattern and current pen size.

Stack Before Call

previous contents	
RgnHandle	Handle to region
	<-SP

Stack After Call

previous contents	
	<-SP

Only points entirely inside the region are affected.

If a region is open and being formed, the outside outline of the region being framed is added to that region's boundary.

Possible Errors

Memory Mgr Errors If a region is open, FrameRgn commands contribute to the definition. Memory manager errors can occur at this time.

PaintRgn Paints (fills) the interior of the specified region with the current pen pattern.

Stack Before Call

previous contents	
RgnHandle	Handle to region
	<-SP

```
Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

EraseRgn Fills the interior of the specified region with the background pattern.

```
Stack Before Call
    |  previous contents    |
    |    RgnHandle          |  Handle to region
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

InvertRgn Inverts the pixels in the interior of the specified region.

```
Stack Before Call
    |  previous contents    |
    |    RgnHandle          |  Handle to region
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

FillRgn Fills the interior of the specified region with the specfied pattern.

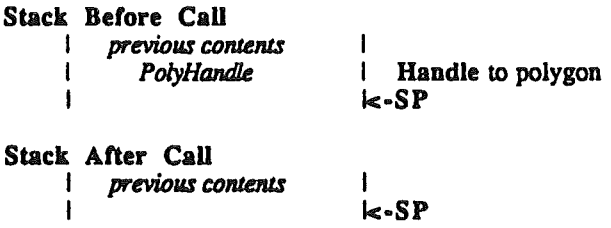
```
Stack Before Call
    |  previous contents    |
    |    RgnHandle          |  Handle to region
    |    PatternPtr         |  Pointer to pattern
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

Drawing Polygons

FramePoly Frames the specified polygon.

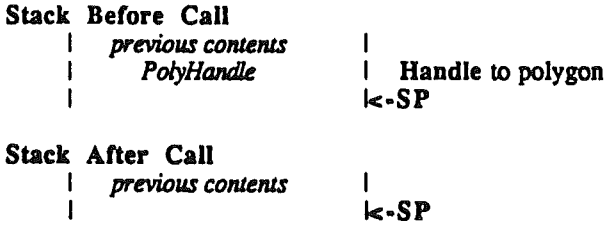


The polygon is framed with a series of lineto calls.

Possible Errors

Memory Mgr Errors If a region is open, FramePoly commands contribute to the definition. Memory manager errors can occur at this time.

PaintPoly Paints the specified polygon.

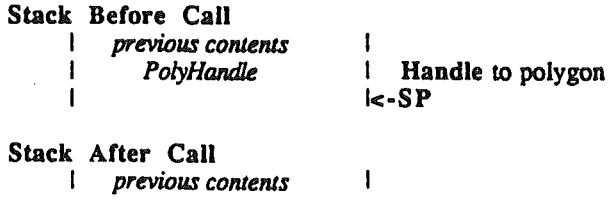


The polygon is painted by opening a region, drawing lines, closing the region and painting the region. When the drawing is complete, the region is discarded. There must be enough memory around to hold the polygon.

Possible Errors

Memory Mgr Errors Any errors from the memory manager are returned unchanged.

ErasePoly Erases the specified polygon.



```
|                          |<-SP
```

The polygon is erased by opening a region, drawing lines, closing the region and erasing the region. When the drawing is complete, the region is discarded. There must be enough memory around to hold the polygon.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

InvertPoly Inverts the specified polygon.

Stack Before Call

```
|   previous contents      |
|     PolyHandle           |  Handle to polygon
|                          |<-SP
```

Stack After Call

```
|   previous contents      |
|                          |<-SP
```

The polygon is inverted by opening a region, drawing lines, closing the region and inverting the region. When the drawing is complete, the region is discarded. There must be enough memory around to hold the polygon.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

FillPoly Paints the specified polygon.

Stack Before Call

```
|   previous contents      |
|     PolyHandle           |  Handle to polygon
|     PatternPtr           |  Pointer to pattern
|                          |<-SP
```

Stack After Call

```
|   previous contents      |
|                          |<-SP
```

The polygon is painted by opening a region, drawing lines, closing the region and painting the region. When the drawing is complete, the region is discarded. There must be enough memory around to hold the polygon.

Possible Errors

Memory Mgr Errors — Any errors from the memory manager are returned unchanged.

Oval Drawing Calls

The code that draws ovals resides in RAM. The code that handles the ovals contribution to region definitions resides in ROM.

FrameOval — Draws the boundary of the oval enscribed in the specified rectangle with the current pattern and pen size.

```
Stack Before Call
    |  previous contents    |
    |      RectPtr          |  Pointer to rectangle.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Only points entirely within the rectangle are affected.

Possible Errors

Memory Mgr Errors — If a region is open, FrameOval commands contribute to the definition. Memory manager errors can occur at this time.

PaintOval — Paints (fills) the interior of the oval enscribed in the specified rectangle with the current pen pattern.

```
Stack Before Call
    |  previous contents    |
    |      RectPtr          |  Pointer to rectangle.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

EraseOval — Erases the interior of the oval enscribed in the specified rectangle with the background pattern.

```
Stack Before Call
    |  previous contents    |
    |      RectPtr          |  Pointer to rectangle.
    |                       |<-SP
```

Stack After Call

previous contents	
	<-SP

Possible Errors

None

InvertOval Inverts the pixels in the interior of the oval enscribed in the specified rectangle.

Stack Before Call

previous contents	
RectPtr	Pointer to rectangle.
	<-SP

Stack After Call

previous contents	
	<-SP

Possible Errors

None

FillOval Paints (fills) the interior of the oval enscribed in the specified rectangle with the specified pattern.

Stack Before Call

previous contents	
RectPtr	Pointer to rectangle.
PatternPtr	Pointer to pattern.
	<-SP

Stack After Call

previous contents	
	<-SP

Possible Errors

None

RoundRect Drawing Calls

The code that draws roundrects resides in RAM. The code that handles the roundrects contribution to region definitions resides in ROM.

FrameRRect Draws the boundary of the roundrect enscribed in the specified rectangle with the current pattern and pen size.

Stack Before Call

previous contents	
RectPtr	Pointer to rectangle.
OvalWidth	integer
OvalHeight	integer
	<-SP

```
Stack After Call
|  previous contents  |
|                     |<-SP
```

Only points entirely within the rectangle are affected.

Possible Errors

Memory Mgr Errors — If a region is open, FrameRRect commands contribute to the definition. Memory manager errors can occur at this time.

PaintRRect — Paints (fills) the interior of the roundrect enscribed in the specified rectangle with the current pen pattern.

```
Stack Before Call
|  previous contents  |
|     RectPtr         |  Pointer to rectangle.
|     OvalWidth       |  integer
|     OvalHeight      |  integer
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

EraseRRect — Erases the interior of the roundrect enscribed in the specified rectangle with the background pattern.

```
Stack Before Call
|  previous contents  |
|     RectPtr         |  Pointer to rectangle.
|     OvalWidth       |  integer
|     OvalHeight      |  integer
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

InvertRRect — Inverts the pixels in the interior of the roundrect enscribed in the specified rectangle.

```
Stack Before Call
|  previous contents  |
|     RectPtr         |  Pointer to rectangle.
|     OvalWidth       |  integer
|     OvalHeight      |  integer
|                     |<-SP
```

```
Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

FillRRect Paints (fills) the interior of the roundrect enscribed in the specified rectangle with the specified pattern.

```
Stack Before Call
    |  previous contents    |
    |     RectPtr           |  Pointer to rectangle.
    |    OvalWidth          |  integer
    |    OvalHeight         |  integer
    |    PatternPtr         |  Pointer to pattern.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

Arc Drawing Calls

Arcs are defined by a rectangle, a start angle (in degrees) and an arc angle (in degrees). The arc that is drawn is the part of the oval described by the rectangle starting at start angle and sweeping clockwise arc angle degrees.

Framed arcs do not contribute to region definitions.

The code that draws ovals resides in RAM.

FrameArc Draws the boundary of the arc enscribed in the specified rectangle with the current pattern and pen size.

```
Stack Before Call
    |  previous contents    |
    |     RectPtr           |  Pointer to rectangle.
    |    StartAngle         |  integer
    |    ArcAngle           |  integer
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Only points entirely within the rectangle are affected.

Possible Errors

None

PaintArc Paints (fills) the interior of the arc enscribed in the specified rectangle with the current pen pattern.

```
Stack Before Call
     |  previous contents    |
     |      RectPtr          |  Pointer to rectangle.
     |    StartAngle         |  integer
     |     ArcAngle          |  integer
     |                       |<-SP

Stack After Call
     |  previous contents    |
     |                       |<-SP
```

Possible Errors

None

EraseArc Erases the interior of the arc enscribed in the specified rectangle with the background pattern.

```
Stack Before Call
     |  previous contents    |
     |      RectPtr          |  Pointer to rectangle.
     |    StartAngle         |  integer
     |     ArcAngle          |  integer
     |                       |<-SP

Stack After Call
     |  previous contents    |
     |                       |<-SP
```

Possible Errors

None

InvertArc Inverts the pixels in the interior of the arc enscribed in the specified rectangle.

```
Stack Before Call
     |  previous contents    |
     |      RectPtr          |  Pointer to rectangle.
     |    StartAngle         |  integer
     |     ArcAngle          |  integer
     |                       |<-SP

Stack After Call
     |  previous contents    |
     |                       |<-SP
```

Possible Errors

None

FillArc Paints (fills) the interior of the arc enscribed in the specified rectangle with the specified pattern.

```
Stack Before Call
    |  previous contents  |
    |      RectPtr        |  Pointer to rectangle.
    |    StartAngle       |  integer
    |     ArcAngle        |  integer
    |     PatternPtr      |  Pointer to pattern.
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

Possible Errors

None

Pixel Transfer Calls

ScrollRect Shifts the pixels inside the intersection of the specified rectangle, VisRgn, ClipRgn, PortRect, and BoundsRect.

```
Stack Before Call
    |  previous contents  |
    |      RectPtr        |  Pointer to rect.
    |         dH          |  integer
    |         dV          |  integer
    |   UpdateRgnHandle   |  Handle
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

The pixels are shifted a distance of *dh* horizontally and *dv* vertically. The positive directions are to the right and down. No other pixels are affected. Pixels shifted out of the scroll area are lost. The backgound pattern fills the space created by the scroll. In addition *UpdateRgn* is changed to the area filled with **BackPat.**

Note that this *UpdateRgn* must be an existing region; it is not created by **ScrollRect.**

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

PaintPixels Transfers a region of pixels.

```
Stack Before Call
    |   previous contents     |
    |    PaintParamPtr        |  Pointer param block.
    |                         |<-SP

Stack After Call
    |   previous contents     |
    |                         |<-SP
```

PaintParamPtr is equal to the following:

PtrToSourceLocInfo	Pointer
PtrToDestLocInfo	Pointer
PtrToSourceRect	Pointer
PtrToDestRect	Pointer
Mode	WORD
MaskHandle (ClipRgn)	Handle

The pixels are transferred without referencing the current GrafPort. The source and destination are described in the input, as is the clipping region.

A DestRect is required, but only the upper left corner is used in this version. In future versions we hope to be able to provide stretching and shrinking. The DestRect shoudl be the same size as the SrcRect for the code to work without changes in the future.

Possible Errors

NotEqualChunkiness	This is returned when the source and destination pixel maps are not the same type. That is one is for a 320 mode display and one is for a 640 mode display. The type is determined by the PortSCB of the LocInfo record.

PPToPort Transfers pixels from a source pixel map to the current port clipping to the current Vis and Clip regions.

```
Stack Before Call
    |   previous contents     |
    |     SrcLocPtr           |  Pointer param block.
    |     SrcRect             |  Pointer source rectangle.
    |     DestX               |  INTEGER.
    |     DestY               |  INTEGER.
    |   Transfer Mode         |  WORD.
    |                         |<-SP

Stack After Call
```

```
|  previous contents  |
|                     |<-SP
```

This call differs form PaintPixels in that the current GrafPort is used as the destination. This is very useful when tring to paint a pixel map to a window and getting the clipping right. PaintPixels can do this but it requires extra code on the application's part.

Possible Errors

NotEqualChunkiness	This is returned when the source and destination pixel maps are not the same type. That is one is for a 320 mode display and one is for a 640 mode display. The type is determined by the PortSCB of the LocInfo record.

Text Drawing and Measuring

All text is drawn in the current font starting at the current pen position using the current textmode and foreground and background colors. After text is drawn, the pen is advanced by the width of the text drawn. The amount of advance can be obtained from any of the width calls. The text box calls return a rectangle that is the bounding box for the text. This width of the bounding box is not necessarily the width of the characters drawn because of kerning. See the appendix on fonts for information on how this works.

DrawChar Draws the spedified character.

```
Stack Before Call
|  previous contents  |
|       theChar       |  word
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

DrawText Draws the spedified text.

```
Stack Before Call
|  previous contents  |
|       TextPtr       |  Pointer
|       TextLen       |  integer
|                     |<-SP

Stack After Call
```

```
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

DrawString Draws the spedified string.

```
Stack Before Call
|  previous contents  |
|     StringPtr       |  Pointer to port.
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

DrawCString Draws the spedified C-String.

```
Stack Before Call
|  previous contents  |
|     StringPtr       |  Pointer to port.
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None

Width Calls

Return the distance the pen would be advanced if the character or sequence of character were to be drawn. Reflects current style, chExtra, spExtra, etc. The actual pen position is not changed.

CharWidth Returns the width of the spedified character.

```
Stack Before Call
|  previous contents  |
|  Space for width    |  Space for integer
|     theChar         |  word
```

```
|                          |<-SP
```

Stack After Call

```
|   previous contents      |
|         width            |  integer
|                          |<-SP
```

Possible Errors

None

TextWidth Returns the width of the spedified text.

Stack Before Call

```
|   previous contents      |
|   Space for width        |  Space for integer
|       TextPtr            |  Pointer
|       TextLen            |  integer
|                          |<-SP
```

Stack After Call

```
|   previous contents      |
|         width            |  integer
|                          |<-SP
```

Possible Errors

None

StringWidth Returns the width of the specified string.

Stack Before Call

```
|   previous contents      |
|   Space for width        |  Space for integer
|       StringPtr          |  Pointer
|                          |<-SP
```

Stack After Call

```
|   previous contents      |
|         width            |  integer
|                          |<-SP
```

Possible Errors

None

CStringWidth Returns the width of the specified C-String.

Stack Before Call

```
|   previous contents      |
|   Space for width        |  Space for integer
|       CStringPtr         |  Pointer
```

```
|                          |<-SP

Stack After Call
|   previous contents      |
|        width             |  integer
|                          |<-SP
```

Possible Errors

None

Bounds Calls

The Bounds calls set the contents of a specified rectangle so that rectangle is the smallest rectangle that encloses all the foreground and background pixels of the specified character or sequence of characters, if they were to be drawn. The rectangle is in local coordinates. It is as tall as the font height, with ascent pixel rows above the current pen position and decent rows below. It extends as far to the left as the current pen position or as far as the leftmost kerning pixel, whichever is further to the left; and as far right as the (hypothetical) new pen position, or the rightmost kerning pixel, whichever is further right. Reflects current style, chExtra, spExtra, etc.

The rectangle is not actually drawn, and the actual pen position is not changed.

CharBounds Sets the specified rectangle to be the bounds of the specified character.

```
Stack Before Call
|   previous contents      |
|        theChar           |  word
|        RectPtr           |  Pointer
|                          |<-SP

Stack After Call
|   previous contents      |
|                          |<-SP
```

Possible Errors

None

TextBounds Sets the specified rectangle to be the bounds of the specified text.

```
Stack Before Call
|   previous contents      |
|        TextPtr           |  Pointer to text.
|        TextLen           |  integer
|        RectPtr           |  Pointer to rect.
|                          |<-SP

Stack After Call
|   previous contents      |
|                          |<-SP
```

Possible Errors

None

StringBounds Sets the specified rectangle to be the bounds of specified string.

```
Stack Before Call
    |  previous contents    |
    |     StringPtr         |  Pointer to string.
    |     RectPtr           |  Pointer to rect.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

CStringBounds Sets the specified rectangle to be the bounds of specified cstring.

```
Stack Before Call
    |  previous contents    |
    |     CStringPtr        |  Pointer to Cstring.
    |     RectPtr           |  Pointer to rect.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None

Various Utilities

Calculations With Rectangles

SetRect Sets the rectangle pointed to by RectPtr to the specified values.

```
Stack Before Call
    |  previous contents  |
    |      RectPtr        |  Pointer
    |        Left         |  integer
    |        Top          |  integer
    |       Right         |  integer
    |       Bottom        |  integer
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

Possible Errors

None.

OffsetRect Offsets the rectangle pointed to by RectPtr by the specified displacements.

```
Stack Before Call
    |  previous contents  |
    |      RectPtr        |  Pointer
    |        dH           |  integer
    |        dV           |  integer
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

dv is added to the top and bottom; *dh* is added to the left and right.

Possible Errors

None.

InsetRect Insets the rectangle pointed to by RectPtr by the specified displacements.

```
Stack Before Call
    |  previous contents  |
    |      RectPtr        |  Pointer
    |        dH           |  integer
    |        dV           |  integer
```

```
|                          |<-SP

Stack After Call
|  previous contents       |
|                          |<-SP
```

dv is added to the top and subtracted from the bottom; *dh* is added to the left and subtracted from the right.

Possible Errors

None.

SectRect Calculates the intersection of two rectangles and places the intersection in a third rectangle.

```
Stack Before Call
|  previous contents       |
|   space for result       |  Space for Boolean empty flag
|      RectPtr             |  Pointer
|      RectPtr             |  Pointer
|      RectPtr             |  Pointer
|                          |<-SP

Stack After Call
|  previous contents       |
|    Empty Flag            |  Boolean (word).
|                          |<-SP
```

If the result is non-empty, the output is TRUE; if the result is empty, the output is FALSE.

Possible Errors

None.

UnionRect Calculates the union of two rectangles and places the union in a third rectangle.

```
Stack Before Call
|  previous contents       |
|      RectPtr             |  Pointer to source rect.
|      RectPtr             |  Pointer to source rect.
|      RectPtr             |  Pointer to destination rect.
|                          |<-SP

Stack After Call
|  previous contents       |
|                          |<-SP
```

Possible Errors

None.

PtInRect Detects whether a specified point is in a specified rectangle.

```
Stack Before Call
    |  previous contents  |
    |  Space for Result   |  Space for Boolean result
    |     PointPtr        |  Pointer
    |     RectPtr         |  Pointer
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |    Result Flag      |  Boolean (word) result
    |                     |<-SP
```

For example:
PtInRect((10,10)),((10,10,20,20)) is TRUE but
PtInRect((20,20)),((10,10,20,20)) is FALSE.

Possible Errors

None.

Pt2Rect Copies one point to the upper left of a specified rectangle and another point to the lower right of the rectangle.

```
Stack Before Call
    |  previous contents  |
    |     Point1Ptr       |  Pointer to source point.
    |     Point2Ptr       |  Pointer to source point.
    |     RectPtr         |  Pointer to destination rect.
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

Possible Errors

None.

EqualRect Compares two rectangles and returns TRUE or FALSE.

```
Stack Before Call
    |  previous contents  |
    |  Space for Result   |  Space for Boolaean (word) result.
    |     Rect1Ptr        |  Pointer to one rect.
    |     Rect2Ptr        |  Pointer to other rect.
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |   Boolean result    |  Boolaean (word) result.
    |                     |<-SP
```

Possible Errors

None.

EmptyRect Returns whether or not a specified rectangle is empty.

Stack Before Call

previous contents	
Space for Result	Space for Boolean (word) result
RectPtr	Pointer
	<-SP

Stack After Call

previous contents	
Result Flag	Boolean (word) result
	<-SP

An empty rectangle has the top greater than or equal to the bottom, or the left greater than or equal to the right.

Possible Errors

None.

Calculations With Points

AddPt Adds two specified points together and leaves the result in the destination point.

Stack Before Call

previous contents	
SrcPtPtr	Pointer to point.
DestPtPtr	Pointer to point used as source and destination
	<-SP

Stack After Call

previous contents	
	<-SP

Possible Errors

None.

SubPt Subtracts the source point from the destination point and leaves the result in the destination point.

Stack Before Call

previous contents	
SrcPtPtr	Pointer to point.
DestPtPtr	Pointer to point used as source and destination
	<-SP

```
Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None.

SetPt Sets a point to specified horizontal and vertical values.

```
Stack Before Call
    |  previous contents    |
    |     SrcPtPtr          |  Pointer to point.
    |     H                 |  Horizontal value of point.
    |     V                 |  Vertical value of point.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Possible Errors

None.

EqualPt Returns a boolean result indicating whether two points are equal.

```
Stack Before Call
    |  previous contents    |
    |  Boolean Result       |  Space for result.
    |     SrcPtPtr          |  Pointer to point.
    |     DestPtPtr         |  Pointer to point used as source and destination
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |  Boolean Result       |  result.
    |                       |<-SP
```

Possible Errors

None.

LocalToGlobal Converts a point from local coordinates to global coordinates.

```
Stack Before Call
    |  previous contents    |
    |     PointPtr          |  Pointer to point.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

Local coordinates are based on the current BoundsRect of the GrafPort. Global coordinates have 0,0 as the upper left corner of the pixel image.

Possible Errors

None.

GlobalToLocal Converts a point from global coordinates to local coordinates.

```
Stack Before Call
    |   previous contents     |
    |       PointPtr          |  Pointer to point.
    |                         |<-SP

Stack After Call
    |   previous contents     |
    |                         |<-SP
```

Local coordinates are based on the current BoundsRect of the GrafPort. Global coordinates have 0,0 as the upper left corner of the pixel image.

Possible Errors

None.

Calculations With Regions

NewRgn Allocates space for a new region and initializes it to the empty region.

```
Stack Before Call
    |   previous contents     |
    |   Space for Handle      |  Space for resulting Handle
    |                         |<-SP

Stack After Call
    |   previous contents     |
    |  Handle to New Rgn      |  Resulting Handle.
    |                         |<-SP
```

This is the only way to create a new region. All other calls work with existing regions.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

DisposeRgn Deallocates space for the specified region.

```
Stack Before Call
    |   previous contents     |
```

```
|     RgnHandle      | Handle to Region being disposed.
|                    |<-SP

Stack After Call
|  previous contents |
|                    |<-SP
```

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

CopyRgn Copies the contents of a region from one region to another.

```
Stack Before Call
|  previous contents |
|    SrcRgnHandle    | Handle to source region.
|    DestRgnHandle   | Handle to destination region
|                    |<-SP

Stack After Call
|  previous contents |
|                    |<-SP
```

If the regions are not the same size to start with, the *DestRgn* is resized. (*DestRgn* must already exist. This call does not allocate it.)

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

SetEmptyRgn Destroys the previous region information by setting it to the empty region.

```
Stack Before Call
|  previous contents |
|     RgnHandle      | Handle to Region being modified
|                    |<-SP

Stack After Call
|  previous contents |
|                    |<-SP
```

The empty region is a rectangular region with a bounding box of (0,0,0,0). If the original region was not rectangular, the region is resized.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

SetRectRgn Destroys the previous region information by setting it to a rectangle described by the input.

```
Stack Before Call
    |  previous contents  |
    |     RgnHandle       |  Handle to region being set.
    |       Left          |  integer
    |       Top           |  integer
    |       Right         |  integer
    |       Bottom        |  integer
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

If the inputs do not describe a valid rectangle, the region is set to the empty region. If the original region was not rectangular, the region is resized.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

RectRgn Destroys the previous region information by setting it to a rectangle described by the input.

```
Stack Before Call
    |  previous contents  |
    |     RgnHandle       |  Handle to region being set.
    |      RectPtr        |  Pointer to rectangle used as source
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

If the input does not describe a valid rectangle, the region is set to the empty region. If the original region was not rectangular, the region is resized.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

OpenRgn Tells QuickDraw II to allocate temporary space and start saving lines and framed shapes for later processing as a region definition.

No Inputs.

This call takes no inputs. Instead it allocates memory to hold information about the being created. When CloseRgn is called, the region is created and this memory is freed.

While the region is open, all calls to **Line, LineTo, FrameRect, FrameOval, FrameRRect, FrameRgn** and **FramePoly** contribute to the region definitition.

Possible Errors

RgnAlreadyOpen	This is returned if you are already saving to a region in this grafport.
Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

CloseRgn Tells QuickDraw II to stop processing information and to return the region that has been created.

```
Stack Before Call
    |  previous contents  |
    |     RgnHandle       |  Handle to region being set to collection of points
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

The region must already exist, and its contents are replaced with the new region.

Possible Errors

RgnNotOpen	This is returned when there is not a region open in the current grafport.
Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

OffsetRgn Moves the region on the coordinate plane a distance of *dh* horizontally and *dv* vertically.

```
Stack Before Call
    |  previous contents  |
    |     RgnHandle       |  Handle to region being offset.
    |        dH           |  Horizontal displacement.
    |        dV           |  Vertical displacement.
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

The region retains its size and shape.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

InsetRgn Shrinks or expands a region.

```
Stack Before Call
    |  previous contents  |
    |     RgnHandle       |  Handle to region being inset.
    |        dH           |  Horizontal displacement.
    |        dV           |  Vertical displacement.
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

All points on the region boundary are moved inwards a distance of *dv* vertically and *dh* horizontally. If *dv* or *dh* are negative, the points are moved outwards in that direction. **InsetRgn** leaves the region "centered" on the same position, but moves the outline. **InsetRgn** of a rectangular region works just like **InsetRect**.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

SectRgn Calculates the intersection of two regions and places the intersection in the third region.

```
Stack Before Call
    |  previous contents  |
    |     RgnHandle1      |  Handle to one source region
    |     RgnHandle2      |  Handle to another source region
    |    DestRgnHandle    |  Handle to destination region
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

The destination region must already exist. The function does not allocate it. However, the destination region may be one of the source regions.

If the regions do not intersect, or one of the regions is empty, the destination is set to the empty region.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

UnionRgn Calculates the union of two regions and places the union in the third region.

```
Stack Before Call
     |  previous contents  |
     |    RgnHandle1       |  Handle to one source region
     |    RgnHandle2       |  Handle to another source region
     |   DestRgnHandle     |  Handle to destination region
     |                     |<-SP

Stack After Call
     |  previous contents  |
     |                     |<-SP
```

The destination region must already exist. The function does not allocate it. However, the destination region may be one of the source regions.

If both regions are empty, the destination is set to the empty region.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

DiffRgn Calculates the difference of two regions and places the difference in the third region.

```
Stack Before Call
     |  previous contents  |
     |    RgnHandle1       |  Handle to one source region
     |    RgnHandle2       |  Handle to another source region
     |   DestRgnHandle     |  Handle to destination region
     |                     |<-SP

Stack After Call
     |  previous contents  |
     |                     |<-SP
```

The destination region must already exist. The function does not allocate it. However, the destination region may be one of the source regions.

If the source region is empty, the destination is set to the empty region.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

XorRgn Calculates the difference between the union and the intersection of two regions and places the result in the third region.

```
Stack Before Call
    |  previous contents  |
    |    RgnHandle1       |  Handle to one source region
    |    RgnHandle2       |  Handle to another source region
    |   DestRgnHandle     |  Handle to destination region
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |                     |<-SP
```

The destination region must already exist. The function does not allocate it. However, the destination region may be one of the source regions.

If the regions are not coincident, the destination is set to the empty region.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

PtInRgn Checks to see whether the pixel below and to the right of the point is within the specified region.

```
Stack Before Call
    |  previous contents  |
    |  Space for Result   |  Space for Boolean (word)
    |     PointPtr        |  Pointer to point.
    |     RgnHandle       |  Region Handle
    |                     |<-SP

Stack After Call
    |  previous contents  |
    |   Boolean Result    |  Boolean (word)
    |                     |<-SP
```

The function returns TRUE if the pixel is within the region and FALSE if it is not.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

RectInRgn Checks whether a given rectangle intersects a specified region.

```
Stack Before Call
```

```
|  previous contents  |
|  Space for Result   |  Space for Boolean (word)
|     RectPtr         |  Pointer to rect.
|     RgnHandle       |  Region Handle
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
|  Boolean Result     |  Boolean (word)
|                     |<-SP
```

The function returns TRUE if the intersection encloses at least one pixel or FALSE if it does not.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

EqualRgn Compares the two regions and returns TRUE if they are equal or FALSE if not.

Stack Before Call

```
|  previous contents  |
|  Space for Result   |  Space for Boolean (word)
|     RgnHandle1      |  Pointer to point.
|     RgnHandle2      |  Region Handle
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
|  Boolean Result     |  Boolean (word)
|                     |<-SP
```

The two regions must have identical sizes, shapes and locations to be considered equal. Any two empty regions are always equal.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

EmptyRgn Checks to see if a region is empty.

Stack Before Call

```
|  previous contents  |
|  Space for Result   |  Space for Boolean (word)
|     RgnHandle       |  Region Handle
|                     |<-SP
```

Stack After Call

```
|  previous contents  |
|  Boolean Result     |  Boolean (word)
|                     |<-SP
```

Returns TRUE if the region is empty or FALSE if not.

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

Calculations with Polygons

OpenPoly Returns a handle to a polygon data structure that will be updated by future LineTo's.

```
Stack Before Call
     |   previous contents     |
     | Space for PolyHandle    |  Space for Handle to Polygon
     |                         |<-SP

Stack After Call
     |   previous contents     |
     |     PolyHandle          |  Handle to Polygon
     |                         |<-SP
```

The polygon is completed by making a ClosePoly call.

Possible Errors

PolyAlreadyOpen	This is returned when a polygon is already open and being saved in the current grafport.
Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

ClosePoly Completes the polygon creation process started with OpenPoly. Has no inputs or outputs.

Possible Errors

PolyNotOpen	This is returned when a polygon is not open in the current grafport.
Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

KillPoly Disposes of the specified polygon.

```
Stack Before Call
     |   previous contents     |
     |     PolyHandle          |  Handle to polygon
     |                         |<-SP

Stack After Call
     |   previous contents     |
     |                         |<-SP
```

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

OffsetPoly Offsets the specified polygon by dH and dV.

```
Stack Before Call
    |   previous contents   |
    |     PolyHandle        |  Handle to polygon.
    |        dH             |  Horizontal displacement
    |        dV             |  Vertical displacement
    |                       |<-SP

Stack After Call
    |   previous contents   |
    |                       |<-SP
```

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

Mapping and Scaling Utilities

MapPt Maps the specified point from the source rect to the dest rect.

```
Stack Before Call
    |   previous contents   |
    |      PointPtr         |  Pointer to point.
    |     SrcRectPtr        |  Pointer to source rect
    |     DestRectPtr       |  Pointer to dest rect
    |                       |<-SP

Stack After Call
    |   previous contents   |
    |                       |<-SP
```

Possible Errors

None.

MapRect Maps the specified rectangle from the source rect to the dest rect.

```
Stack Before Call
    |   previous contents   |
    |       RectPtr         |  Pointer to rectangle.
    |     SrcRectPtr        |  Pointer to source rect
    |     DestRectPtr       |  Pointer to dest rect
    |                       |<-SP

Stack After Call
```

```
|  previous contents  |
|                     |<-SP
```

Possible Errors

None.

MapRgn Maps the specified region from the source rect to the dest rect.

```
Stack Before Call
|  previous contents  |
|     MapRgn          |  Handle to region.
|     SrcRectPtr      |  Pointer to source rect
|     DestRectPtr     |  Pointer to dest rect
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

MapPoly Maps the specified polygon from the source rect to the dest rect.

```
Stack Before Call
|  previous contents  |
|     PolyHandle      |  Handle to polygon.
|     SrcRectPtr      |  Pointer to source rect
|     DestRectPtr     |  Pointer to dest rect
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None.

ScalePt Scales the specified point from the source rect to the dest rect.

```
Stack Before Call
|  previous contents  |
|     PointPtr        |  Pointer to point.
|     SrcRectPtr      |  Pointer to source rect
|     DestRectPtr     |  Pointer to dest rect
|                     |<-SP

Stack After Call
|  previous contents  |
|                     |<-SP
```

Possible Errors

None.

Miscellaneous Utilities

Random Returns a pseudorandom number in the range –32768 to 32767.

```
Stack Before Call
    |  previous contents    |
    |  Space for integer    |  Space for returned integer
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |  Random integer       |  integer
    |                       |<-SP
```

The number returned is generated based upon calculations performed on *SeedValue*, which can be set with **SetRandSeed**. The result for any particular seed value is always the same.

Possible Errors

None.

SetRandSeed Sets the seed value for the random number generator.

```
Stack Before Call
    |  previous contents    |
    |    RandomSeed         |  long integer
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |                       |<-SP
```

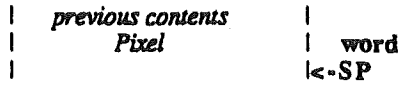
The algorithm for random numbers uses a 32 bit seed to produce a 16 bit random number.

Possible Errors

None.

GetPixel Returns the pixel below and to the right of the specifed point.

```
Stack Before Call
    |  previous contents    |
    |  Space for Pixel      |  Space for word
    |        H              |  Horizontal value of point.
    |        V              |  Vertical value of point.
    |                       |<-SP

Stack After Call
```



ThePixel is returned in the lower bits of the word. If the current drawing location has a chunkiness of 2, then 2 bits of the word are valid. If the current drawing location has a chunkiness of 4, then 4 bits of the word are valid.

There is no guarantee that the point actually belongs to the port.

Possible Errors

None.

Customizing QuickDraw Operations

These routines work similarly to those in QuickDraw on the Macintosh. The major difference is that no inputs are passed on the stack. Instead the standard routines expect their inputs on zero page at particular locations. Moreover, they expect that QuickDraw's zero page is already switched in when they are called. Details on what parts of zero page are used for what are not available yet.

A final difference is how these routines are called. Rather than making standard tool calls, you access them through vectors in bank E0.

SetStdProcs Sets up the specified record of pointers.

```
Stack Before Call
     |   previous contents       |
     |Pointer to StdProc Record  |  Pointer to standard proc record.
     |                           |<-SP

Stack After Call
     |   previous contents       |
     |                           |<-SP
```

Possible Errors

None.

StdText Draws standard text.

StdLine Draws standard lines.

StdRect Draws standard rects.

StdRRect Draws standard round rects.

StdOval Draws standard ovals.

StdArc Draws standard arcs.

StdPoly Draws standard polys.

StdRgn Draws standard regions.

StdPixels Draws standard pixels.

StdComment Does standard comments for pictures.

StdTxMeas Does standard text measuring.

StdGetPic Does standard retieval from picture record.

StdPutPic Does standard storage into picture record.

SetIntUse Tells QuickDraw's cursor drawing code whether or not it should use scan line interrupts.

```
Stack Before Call
     |   previous contents     |
     |        UseInt           |  word
     |                         |<-SP

Stack After Call
     |   previous contents     |
     |                         |<-SP
```

QuickDraw normally uses scan line interrupts to draw the cursor without flicker. If an application wants to use scan line interrupts for some process of its own, it must tell QuickDraw not to use them.

GetAddress Returns the address of the specified table.

```
                  input      ID                    WORD
                  output     TablePtr              LONG
Stack Before Call
     |   previous contents     |
     |   Space for Pointer     |  Space for Pointer to table in ROM
     |          ID             |  integer
     |                         |<-SP

Stack After Call
     |   previous contents     |
     |                         |<-SP
```

The ID's supported are

1	ScreenTable
2	ConTable320
3	ConTable640

QuickDraw II contains a number of tables that may be useful to a programmer. The GetAddress call is provided to make these tables accessable. It is absolutely imperitive that a program obtain the address of a table every time it runs. We will make no guarentee that these tables will stay in the same place when we change the ROM. In fact we can guarentee

that these tables will move and that anyone who hard-codes the addresses of these tables in their programs will be sorry.

The screen table has 200 two byte entries. Each is the address of the start of a scan line in the display buffer. The zeroth entry is $2000, the address of scan line zero. Entry 1 is $20A0, the address of scan line one. And so on.

ConTable320 and ConTable640 are used to convert from bytes that are one bit per pixel to bytes that are four and two bits per pixel respectively. ConTable320 has 256 four byte entries while ConTable640 has 256 two byte entries. These entries are the two and four bit per pixel representation of one bit per pixel bytes. For example, the byte containing $37 looks as follows in one, two and four bit per pixel mode.

One Bit	Two Bit	Four Bit
%00110111	%00 00 11 11 00 11 11 11	$00FF 0FFF

The two and four bit versions would be obtained from the table as follows:

```
lda OneBit              lda OneBit              pick up the byte
and #$00FF              and #$00FF              mask off the high byte
asl a                   asl a                   multiply by 2 or 4
tay                     asl a
lda [TwoBitTable],y     tay                     put result in y
                        lda [FourBitTable],y    load out of table through y
                        tax                     (save in x)
                        iny                     (bump y)
                        iny
                        lda [FourBitTable],y    (get the second word)
```

In both cases the addresses obtained from GetAddress are already on zero page.

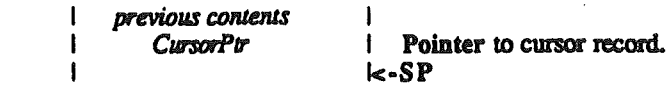
Possible Errors

NotActive	This is returned when QuickDraw has not been initialized.

Cursor-Handling Routines

SetCursor Sets the cursor to the image passed in the cursor record.

Stack Before Call



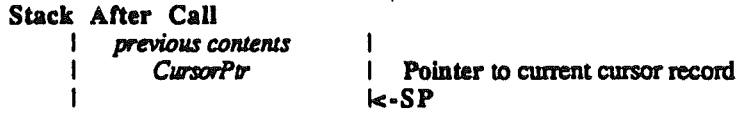
Stack After Call

If the cursor is hidden, it remains hidden and appears in the new form when it becomes visible again. If the cursor is visible, it appears in the new form immediately.

GetCursorAdr Returns a pointer to the current cursor record.

Stack Before Call

Stack After Call



HideCursor Decrements the cursor level. A cursor level of zero indicates the cursor is visible; a cursor level less than zero indicates the cursor is not visible.

No Inputs.

ShowCursor Increments the cursor level unless it is already zero. A cursor level of zero indicates the cursor is visible; a cursor level less than zero indicates the cursor is not visible.

No Inputs.

ObscureCursor Hides the cursor until the mouse moves. This tool is used to get the cursor out of the way of typing.

No Inputs.

InitCursor Reinitializes the cursor.

No Inputs.

The cursor is set to the arrow cursor and made visible. This routine also checks the MasterSCB and sets the cursor accordingly. This is the routine to use if for some reason you want to change modes in the middle of a program. The steps you take to do this are:

1. Hide the cursor if it is not already hidden.
2. Set the MasterSCB to the mode you want.
3. Set all the SCB's to the MasterSCB.
4. Set the color table the way you want it.
5. Repaint the screen for the new mode
6. Call InitCursor.