QuickDraw II
External Reference Specification

Steven E. Glass
Bennet Marks

April 25, 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

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. We do not have enough time to provide all of QuickDraw's capabilities in the ROM so we have a two part approach. The lowest-level routines (I call them the Core Routines) go into ROM. They will provide as much as we can in the given time.

The higher level routines provide the rest of QuickDraw's capabilities. (I don't have a name yet.) The ROM routines combined with these RAM based extensions are what I call **QuickDraw II.**

Appendix C contains a comparison of QuickDraw with QuickDraw II.

Design Goals and Capabilities

The goal for the QuickDraw II core routines is to provide as much as possible in the time we have, but provide at least a minimal set of graphics routines which allow us to

- Build window and menu managers.
- Provide a minimal useful set of graphics routines.
- Allow RAM based code easy access to the lowest level routines so that higher level drawing routines will perform as if they part of the same package.

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

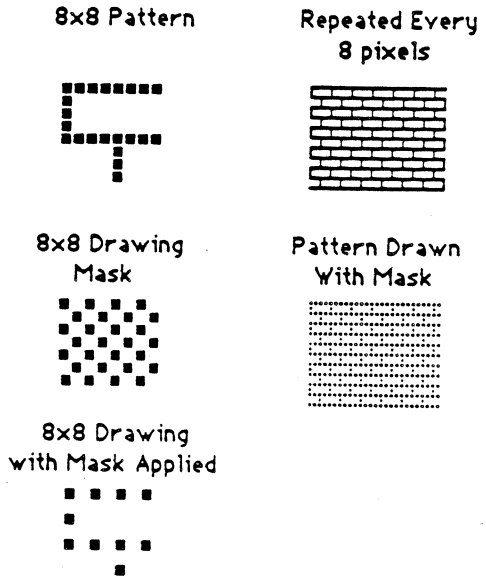
Lines
Rectangles
Regions
Polygons
Pixel Images
Text

Lines, rectangles, regions and polygons 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

[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.

drawn. Only those pixels in the pattern aligned with a "1" in the mask are drawn. See figure 0 below.

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.

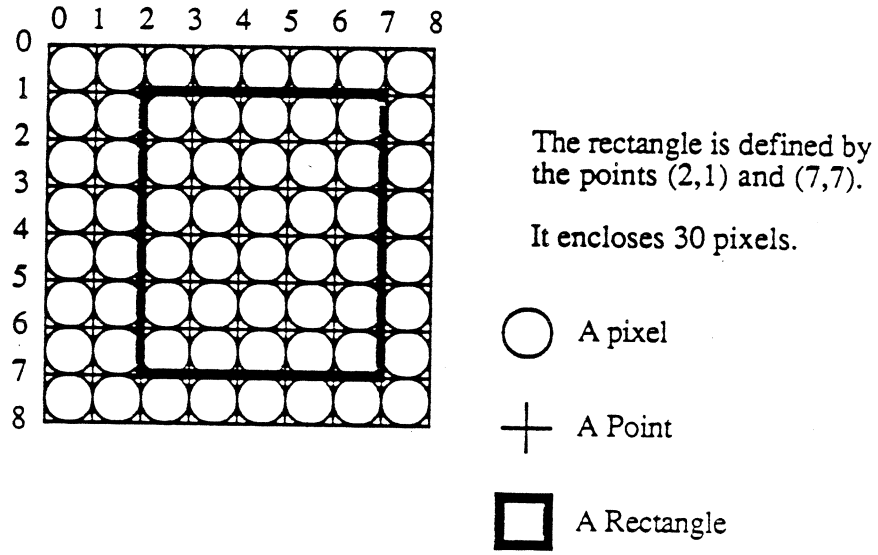
Basic Concepts and Terminology

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.

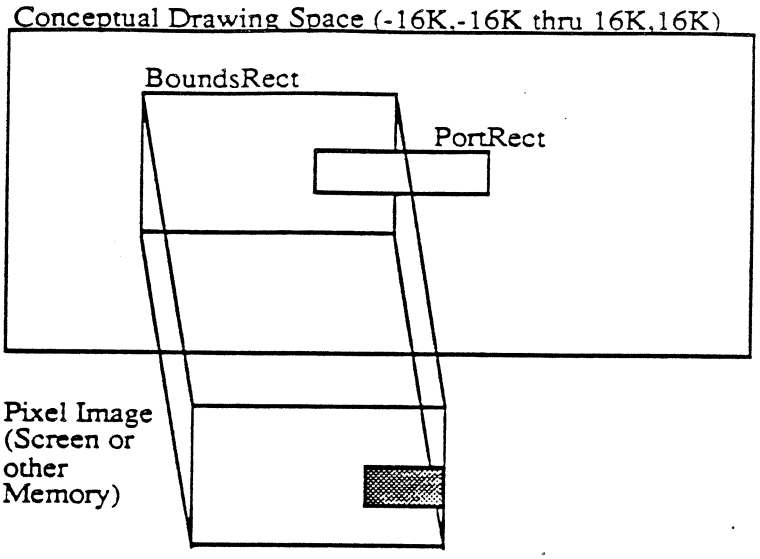
A pixel map need not be the area of memory associated with the graphics screen. QuickDraw II can treat other memory as pixel map memory and draw into it as easily as the screen memory.

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.

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

---

[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.

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

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;

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

GrafPort

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

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

UserField : long
SysField : long

Font

See Appendix D

FontInfoRecord

Ascent	: integer
Descent	: integer
WidMax	: integer
Leading	: 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'0f000000000000000'
        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.

Release Notes for Beta 2.0 Tools ROM

April 24, 1986

The following tools are included in ROM.

Tool Locator

See February 19, 1986 ERS for latest documentation. All functions are available.

Memory Manager

See March 10, 1986 ERS for latest documentation.

Misc. Tools

See March 10, 1986 ERS for latest documentation. All functions are available.

QuickDraw II

See April 24, 1986 ERS for latest documentation.

Errors should be correctly reported.

Polygons have been implemented.

One function is not implemented: GetFontGlobals.

The text routines do not support any style changes or scaling.

The SpaceExtra feature is implemented incorrectly. The current GrafPort only has space for an integer where the field should be a fixed point value.

Desk Manager

The lastest ERS is January 1, 1986, but this is not accurate. A new ERS is about to be published. There is support for RAM-based classic desk accessories.

Event Manager

See March 4, 1986 ERS for latest documentation. All functions are available.

Scheduler

See February 17, 1986 ERS for latest documentation. All functions are available.

Sound Tools

See March 24, 1986 ERS for documentation.

FDP Tool

See March 12, 1986 ERS for documentation. All functions are available.

Integer Math Tools

See April 22, 1986 ERS for latest documentation. The Dec2Int, Int2Dec, Dec2Long and Long2Dec have changed and now take an additional parameter. All functions planned for ROM are available.

Text Tools

The text tools have changed dramatically. See the April 23, 1986 ERS for latest documentation. All functions are available.

SANE

See the December 12, 1985 ERS for documentation. The floating point engine is available and passes all 31,000 test vectors. There is a formatter in ROM but it is untested. The SCAN and elementary functions are not available.

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 two 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.

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 can fail 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.

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

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

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).

```
Stack Before Call
     |   previous contents     |
     |      MasterSCB          |   Word
     |                         |<-SP

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

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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	Red	F 0 0	
2	Green	0 F 0	
3	Blue	0 0 F	
4	Teal	0 8 8	
5	??	8 0 8	
6	Brown	0 6 6	
7	Dark Gray	5 5 5	
8	Light Gray	A A A	
9	Orange	F 8 0	
10	???	8 F 8	
11	???	F 8 8	
12	Yellow	F F 0	
13	Magenta	F 0 F	
14	Cyan	0 F F	
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	Opposite of White
1	Red	F 0 0	
2	Blue	0 0 F	
3	White	F F F	Opposite of Black

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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.

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
```

It will put the handle to this font in the every port opened or inited.

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

GetSysFont Returns a handle to the current system font.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

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

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

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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.

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

HidePen Decrements the pen level. A pen level of zero indicates drawing will occur; a pen level less than zero indicates drawing will not occur.

No Inputs.

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

ShowPen Increments the pen level unless it is already zero. A pen level of zero indicates that drawing will occur; a pen level less than zero indicates drawing will not occur.

No Inputs.

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

GetPen Returns the pen location.

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

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

Possible Errors

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

GetPenMode — Sets the current pen mode to the specified pen mode.

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

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

Possible Errors

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

GetPenMask Returns the pen mask at the specified location.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

SetBackPat Sets the background pattern to the specified pattern.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

GetBackPat Returns the background pattern at the specified location.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

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
```

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

NotActive — This is returned whan QuickDraw has not been initialized.

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
```

Possible Errors

NotActive — This is returned whan QuickDraw has not been initialized.

SetFontFlags — Sets the font flags to the specified value.

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

```
|      FontFlags        |  word
|                       |<-SP
```

Stack After Call

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

SetTextFace Sets the text face to the specified value.

Stack Before Call

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

Stack After Call

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

SetTextMode Sets the text mode to the specified value.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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
```

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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	**word**
	<-SP

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

SetClipHandle 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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

GetVisHandle — Returns a copy of the handle to the VisRgn.

Stack Before Call

previous contents	
Space for Result	Space for **Handle**
	<-SP

Stack After Call

previous contents	
RgnHandle	**Handle**
	<-SP

Possible Errors

NotActive — This is returned whan QuickDraw has not been initialized.

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

previous contents	
PicSaveValue	**Long**
	<-SP

Stack After Call

previous contents	
	<-SP

Possible Errors

NotActive — This is returned whan QuickDraw has not been initialized.

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

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

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

Possible Errors

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive — This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

Stack Before Call

previous contents	
Space for Long	Space for PolySaveValue
	<-SP

Stack After Call

previous contents	
PolySaveValue	**Long**
	<-SP

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

SetGrafProcs Sets the GrafProcs field to the value passed.

Stack Before Call

previous contents	
GrafProcsPtr	**Pointer**
	<-SP

Stack After Call

previous contents	
	<-SP

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

```
Stack Before Call
    |  previous contents    |
    |    Space for Ptr      |  Space for Pointer to Graf Procs Record.
    |                       |<-SP

Stack After Call
    |  previous contents    |
    |    GrafProcsPtr       |  Pointer Graf Procs Record.
    |                       |<-SP
```

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

Drawing Polygons

FramePoly Frames the specified polygon.

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

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

The polygon is framed with a series of lineto calls.

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.
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.

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

NotActive	This is returned whan QuickDraw has not been initialized.
Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

ErasePoly Erases the specified polygon.

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

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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
Memory Mgr Errors	Any errors from the memory manager are returned unchanged.

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

NotActive	This is returned whan QuickDraw has not been initialized.
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
PtrToDes:LocInfo	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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

DrawText

Draws the spedified text.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

DrawString Draws the spedified string.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

DrawCString Draws the spedified C-String.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

CharBounds Fills in the specified rectangle with bounds of character.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

TextBounds Fills in the specified rectangle with bounds of 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

NotActive	This is returned whan QuickDraw has not been initialized.

StringBounds Fills in the specified rectangle with bounds of string.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

CStringBounds Fills in the specified rectangle with bounds of CString.

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

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

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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.

SectRect Calculates the intersection of two rectangles and places the

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
    |         dI            |  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 rect.
    |     Point2Ptr       |  Pointer to source rect.
    |     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     |
    |       PortPtr           |  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

NotActive	This is returned whan QuickDraw has not been initialized.

GlobalToLocal Converts a point from global coordinates to local coordinates.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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.

While the region is open, all calls to **Line, LineTo,** and the procedures that draw framed shapes affect the outline of the region.

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned when QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.
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

NotActive	This is returned whan QuickDraw has not been initialized.

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

NotActive	This is returned whan QuickDraw has not been initialized.

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
```

Possible Errors

NotActive	This is returned whan QuickDraw has not been initialized.

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

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

NotActive	This is returned whan QuickDraw has not been initialized.

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 whan QuickDraw has not been initialized.

Cursor-Handling Routines

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

```
Stack Before Call
|   previous contents   |
|      CursorPtr        |  Pointer to cursor record.
|                       |<-SP

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

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
|   previous contents   |
|   Space for Pointer   |  Space for Pointer current cursor record.
|                       |<-SP

Stack After Call
|   previous contents   |
|      CursorPtr        |  Pointer to current cursor record
|                       |<-SP
```

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.

Transfer Modes

There are eight different pen modes. These modes are used to derive the color of a pixel when it is being drawn to. Each pixel is made up of a series of bits. The pen operates on the individual bits in a pixel as single units. In this way logical binary operations are well defined.

The codes for the various pen modes are different from the codes used in QuickDraw on the Macintosh. Similar modes for text, pen and pixel transfers all use the same codes. The codes for the inverted modes are the same as the orignal mode except that the high bit of the word is set.

The following transfer modes are available. (Each 1 and 0 is the value of a bit in a pixel.)

Mode $0000 (COPY)
$8000 (notCOPY)

Copy SRC (or not SRC) to destination. Copy is the typical drawing mode. For text, the fully colored text pixels (both foreground and background) are copied into the destination.

copy		Pen 0	Pen 1
Dest.	0	0	1
	1	0	1

notcopy		Pen 0	Pen 1
Dest.	0	1	0
	1	1	0

Mode $0001 (OR)
$8001 (notOR)

Overlay (OR) SRC (or not SRC) and destination. You can use this mode to non-destructively overlay new images on top of existing images and its inverse to overlay inverted images. For text, the fully colored text pixels (both foreground and background) are ORed with the destination.

OR		Pen 0	Pen 1
Dest.	0	0	1
	1	1	1

notOR		Pen 0	Pen 1
Dest.	0	1	0
	1	1	1

Mode **$0002 (XOR)**
$8002 (notXOR)

Exclusive or (XOR) pen with destination. You can use this mode and its inversion for cursor drawing and rubber-banding. If an image is drawn in penXOR mode, the appearance of the destination at the image location can be restored merely by drawing the image again in penXOR mode. For text, the fully colored text pixels (both foreground and background) are XORed with the destination.

XOR		Pen 0	Pen 1
Dest.	0	0	1
	1	1	0

notXOR		Pen 0	Pen 1
Dest.	0	1	0
	1	0	1

Mode **$0003 (BIC)**
$8003 (notBIC)

Bit Clear (BIC) pen with destination ((NOT pen) AND destination). You can use this mode to explicitly erase (turn off) pixels, often prior to overlaying another image. notBIC can be used to display the intersection of two images. For text, the fully colored text pixels (both foreground and background) are BICed with the destination.

BIC		Pen 0	Pen 1
Dest.	0	0	0
	1	1	0

notBIC		Pen 0	Pen 1
Dest.	0	0	0
	1	1	0

Special Text Modes. The following modes are only used for text. They apply when drawing from a 1-bit per pixel world to a 2 or 4 bit per pixel world. This only occurs when drawing from the font to a destination pixel map.

Mode $0004 (foreCOPY)
$8004 (notforeCOPY)

Copies only the foreground pixels into the destination. Background pixels are not altered. The inverted mode inverts the foreground pixels before copying them.

Mode $0005 (foreOR)
$8005 (notforeOR)

ORs only the foreground pixels into the destination. Background pixels are not altered. The inverted mode inverts the foreground pixels before the OR operation occurs.

Mode $0006 (foreXOR)
$8006 (notforeXOR)

XORs only the foreground pixels into the destination. Background pixels are not altered. The inverted mode inverts the foreground pixels before the XOR operation occurs.

Mode $0007 (foreBIC)
$8007 (notforeBIC)

BICs only the foreground pixels into the destination. Background pixels are not altered. The inverted mode inverts the foreground pixels before the BIC operation occurs.

Appendix B
Hardware Summary

The Super Hi-Res Graphics hardware can display 200 scan lines and many colors. The following four features are controlled independently for each scan line:

Color Table	One of 16
Fill Mode	On or Off
Interrupt	On or Off
Color Mode	320 vs 640 pixels per scan line

The scan line control byte (SCB) controls these four features for each scan line. The low nibble of the SCB identifies the color table to be used for this scan line. Bit 4 is reserved. Bit 5 of the SCB controls fill mode: 1 is on, 0 is off. Bit 6 of the SCB controls interrupts: if the bit is set then an interrupt will be generated when the scan line is refreshed. Bit 7 of the SCB controls the mode: 0 is 320, 1 is 640.

```
 7 6 5 4  3 2 1 0
|---|---|---|---||---------------
 M I F R     Color Table
```

Color Table

A color table is a table of 16 two byte entries. The low nibble of the low byte is the intensity of the color blue. The high nibble of the low byte is the intensity of the color green. The low nibble of the high byte is the intensity of the color red. The high nibble of the high byte is not used. Pixels in 320 mode are 4 bits wide and their numeric representation identifies a color in the color table. Pixels in 640 mode are two bits wide and their numeric representation identifies a color in a subset of the full color table. The first pixel in the byte (bits 0 and 1) selects one of four colors in the table from 0 thru 3. The second pixel in the byte (bits 2 and 3) selects one of four colors in the table from 4 thru 7. The third pixel in the byte (bits 4 and 5) selects one of four colors in the table from 8 thru 11. The fourth pixel in the byte (bits 6 and 7) selects one of four colors in the table from 12 thru 15.

HighByte		LowByte	
High Nibble	Low Nibble	High Nibble	Low Nibble
Reserved	Red	Green	Blue

Fill Mode

When fill mode is active, the 0th color in the color table becomes inactive. A pixel with a numeric value of zero serves as a place holder indicating that the pixel should be displayed as the same color last displayed.

Scan Line Values

1 0 0 0 0 2 0 0 0 0 0 1 0 0 0 0

Colors Shown

B B B B B W W W W W W B B B B B

Interrupts

Interrupts can be used to synchronize drawing with veritcal blanking so pixels are not changed as they are being drawn (a pixel is drawn once every 1/60 of a second). Interrupts can also be used to change the color table before a screen is completely drawn. This will allow a program to show more than 256 colors on the screen at once (but at the cost of servicing the interrupt).

Appendix C
Comparison to QuickDraw

QuickDraw has the following functional parts. Next to each part I indicate where the part will fall.

Environmental Control	Core
Rectangle Drawing	Core
Line Drawing	Core
Pixel Map Transfer	Core
Region Clipping	Core
Cursor Support and Drawing	Core
Utilities	Core
Text	Core
Region Manipulation	Core
Round Things (ovals, circles, round rects and arcs)	RAM
Pictures	RAM
Polygons	Core
Advanced Routines	Core

Each routine in QuickDraw is listed below with its corresponding QuickDraw II routine. An entry under QuickDraw II of "same" or "similar" means that the routine will work just like or somewhat like the corresponding QuickDraw routine. A minus sign indicates that the routine will not be present. Some entries are the names of calls as they will appear in QuickDraw II (different from QuickDraw). A question mark indicates that we are not yet sure. Finally, An explanation point indicates that prototype code is running today.

QuickDraw Routine	QuickDraw II Core	QuickDraw II RAM
InitGraf	Different!	
OpenPort	same!	
InitPort	same!	
ClosePort	same!	
SetPort	same!	
GetPort	same!	
GrafDevice	-	
SetPortBits	SetPortLoc!	
PortSize	same!	
MovePortTo	same!	
SetOrigin	same!	
SetClip	same!	
GetClip	same!	
ClipRect	same!	
BackPat	same!	
InitCursor	same!	
SetCursor	same!	

HideCursor	same!
ShowCursor	same!
ObscureCursor	same!
HidePen	same!
ShowPen	same!
GetPen	same!
GetPenState	same!
SetPenState	same!
PenSize	same!
PenMode	same!
PenPat	same!
PenNormal	same!
MoveTo	same!
Move	same!
LineTo	same!
Line	same!
TextFont	SetFont!
TextFace	sortof!
TextMode	same!
TextSize	no
SpaceExtra	same!
DrawChar	same!
DrawString	same!
DrawText	same!
CharWidth	same!
StringWidth	same!
TextWidth	same!
GetFontInfo	same!
ForeColor	SetForeColor!
BackColor	SetBackColor!
	GetForeColor!
	GetBackColor!
ColorBit	-
SetRect	same!
OffsetRect	same!
InsetRect	same!
SectRect	same!
UnionRect	same!
PtInRect	same!
Pt2Rect	same!
PtToAngle	?
EqualRect	same!
EmptyRect	same!
FrameRect	same!
PaintRect	same!
EraseRect	same!
InvertRect	same!
FillRect	same!

FrameOval		same
PaintOval		same
EraseOval		same
InvertOval		same
FillOval		same
FrameRoundRect		same
PaintRoundRect		same
EraseRoundRect		same
InvertRoundRect		same
FillRoundRect		same
FrameArc		?
PaintArc		?
EraseArc		?
InverArc		?
FillArc		?
NewRgn	same!	
DisposeRgn	same!	
CopyRgn	same!	
SetEMptyRgn	same!	
SetRectRgn	same!	
RectRgn	same!	
OpenRgn	same!	
CloseRgn	same!	
OffsetRegion	same!	
InsetRgn	same!	
SectRgn	same!	
UnionRgn	same!	
DiffRgn	same!	
XorRgn	same!	
PtInRgn	same!	
RectInRgn	same!	
EqualRgn	same!	
EmptyRgn	same!	
FrameRgn	same!	
PaintRgn	same!	
EraseRgn	same!	
InvertRgn	same!	
FillRgn	same!	
ScrollRect	same!	
CopyBits	PaintPixels!	
OpenPicture		same
PicComment		same
ClosePicture		same
DrawPicture		same
KillPicture		same
OpenPoly	same!	
ClosePoly	same!	

KillPoly	same!
OffsetPoly	same!
FramePoly	same!
PaintPoly	same!
ErasePoly	same!
InvertPoly	same!
FillPoly	same!
AddPt	same!
SubPt	same!
SetPt	same!
EqualPt	same!
LocalToGlobal	same!
GlobalToLocal	same!
Random	same!
GetPixel	similar!
StuffHex	unlikely
ScalePt	same!
MapPt	same!
MapRect	same!
MapRgn	same!
MapPoly	same!
SetStdProcs	same!
StdText	similar!
StdLine	similar!
StdRect	similar!
StdRRect	similar!
StdOval	similar!
StdArc	similar!
StdPoly	similar!
StdRgn	similar!
StdBits	similar!
StdComment	similar!
StdTxMeas	similar!
StdGetPic	similar!
StdPutPic	similar!

Appendix D
Font Definition

Coming some day.

Appendix E
Change History

First Publication June 15, 1985

Second Publication July 10, 1985
- Added summary of hardware.
- Added Global Environment calls.
- Fixed typos reported by Harvey.
- A few GrafPort calls were made more like QuickDraw. QuickDraw names were adopted whereever there were questions.

Third Publication July 25, 1985
- Added commentary about initial review.
- Added Line Drawing calls.
- Raised questions of code size.

Fourth Publication September 17, 1985
- Mode is part of GrafPort
- Patterns are private.
- The bounds rect is private

- PenPat => SetPenPat
- BackPat => SetBackPat
- GetPenPat
- GetBackPat
- Appendix C

Fifth Publication September 25, 1985
- General Clarifications
- Clipping

Sixth Publication October 29, 1985
- General Clarifications
- GrafPort is no longer private

Seventh Publication November 21, 1985
- Adjusted to new tool locator specifications
- Adjusted to new memory manager specifications
- Added section on calling conventions
- Added utility calls to ERS
- Removed low level slab and slice routines and low level clipping routine information from this document.
- SetPenPat => PenPat
- SetBackPat => BackPat

Eighth Publication December 3, 1985
- Refined Color Table
- Removed Scroll Rect from Core Routines (they will be part of the expanded routines now).
- Updated Appendix C

Ninth Publication January 15, 1986

Changed the Name to QuickDraw II
Removed Text Calls from this document.
Added Region calls.
Added ScrollRect
Updated Appendix C

Tenth Publication March 5, 1986

Made GrafPort private and added calls for accessing fields.
Added PenMask to grafport and drawing definition.
Added calls for PenMask.
Redefined codes for transfer modes. Added two text transfer modes.
Added information on the Cursor data structure.
Added information on Customizing QuickDraw operations.
Added Text Calls and appendix on font definition. Added new font field to GrafPort.
Added UserField and System Field to GrafPort.
Added GetAddress Call.
Updated Appendix C.

Eleventh Publication April 4, 1986

Added Polygon calls.
Added SetIntUse call.
Added Get & Set SysFont calls.
Added Get & Set VisRgn calls.

Twelfth Publication April 25, 1986

Reoganized all the calls.
Augmented introductory information
Added missing calls:
- ClearScreen
- GrafOn
- GrafOff
- GetClipHandle
- SetClipHandle
- GetVisHandle
- SetVisHandle
- InitCursor

Changed the way inputs are described.
Changed input to SetRandSeed from integer to long.