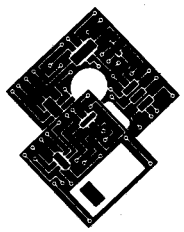
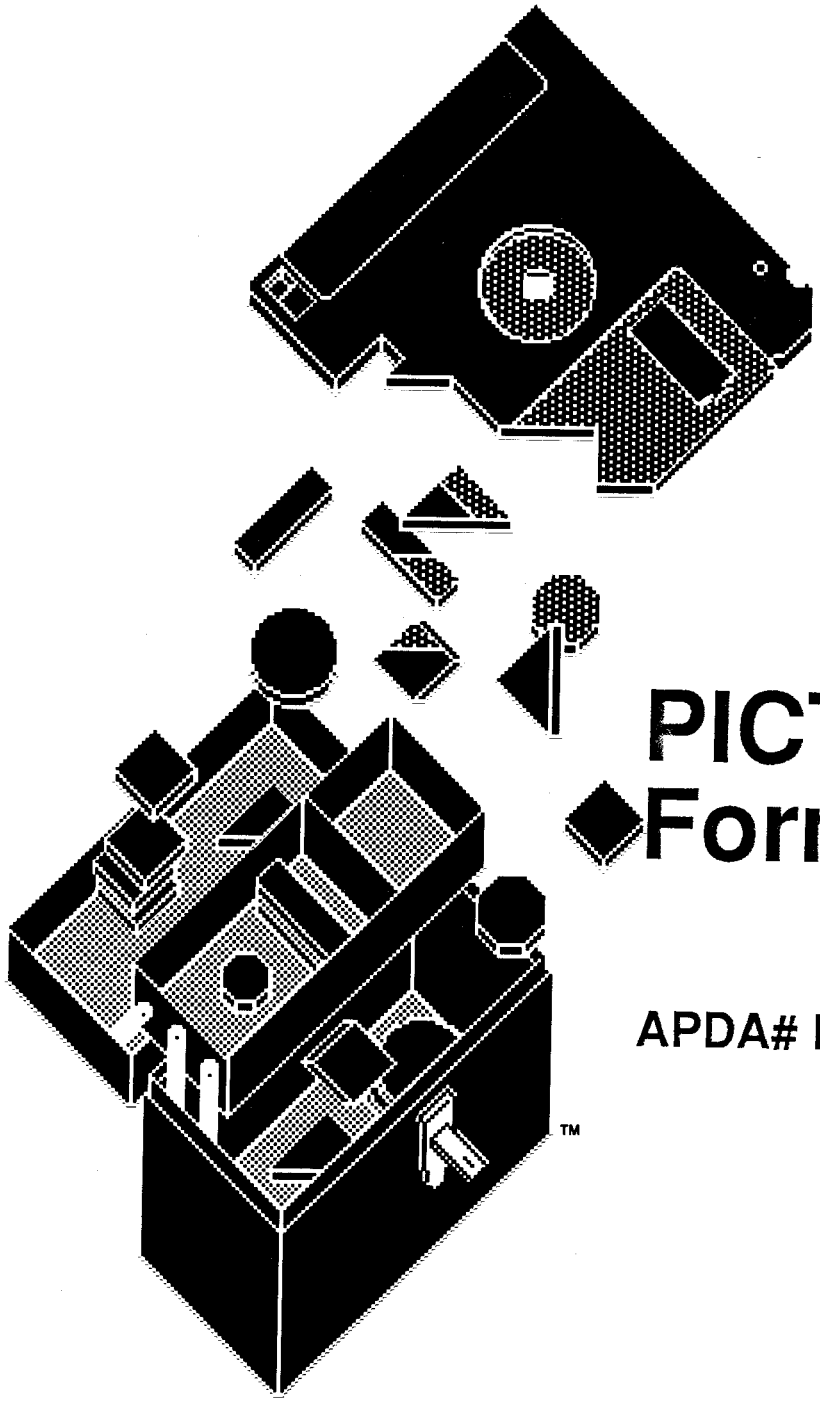


APPLE
PROGRAMMER'S
AND DEVELOPER'S
ASSOCIATION



PICT File Format Notes

APDA# KMBPFN

Macintosh File Format & Picture Structure for Graphic Applications

July 15, 1987

Apple Technical Publications

Contents

Figures & Tables

About these notes

This document, in addition to *Inside Macintosh* and *Macintosh Technical Notes #21 , 27* and *120*, provides the information developers need to know in order to create or modify graphics application programs for the Macintosh. It describes the graphic data file structure, how it is parsed, the format used to define pictures in QuickDraw and Color QuickDraw, and the differences between the new opcodes and those currently in use. It is written with particular consideration for the needs of scanner application developers and manufacturers, but applies in general to all developers of applications that generate or receive image data and do not directly deal with structured graphics.

Apple's main objective in releasing this information is to create a development environment that will ensure compatibility with existing products and spawn new graphics-oriented tools. These preliminary notes do not constitute a manual and should not be considered complete in their present form. While every attempt has been made to verify the accuracy of the information presented, it is subject to change without notice.

Supplemental reference documents

The Apple publications listed below are suggested reading to help understand QuickDraw pictures:

- *Inside Macintosh*, Volumes I & IV contain detailed information about QuickDraw
- *Inside Macintosh*, Volume V (currently in preliminary release); contains detailed information about Color QuickDraw
- Macintosh Technical Notes #21 describes the internal format of the version 1 QuickDraw picture data structure
- Macintosh Technical Notes #27 describes the PICT file format
- Macintosh Technical Notes #120 describes drawing to an offscreen buffer
- Macintosh Technical Notes #154 describes how to display large PICT files

Terminology

The following terms are used throughout this document.

QuickDraw picture:	A variable-sized QuickDraw data structure, consisting of picture opcodes and data, that is used to store graphics primitives for later playback. A version 2 picture can contain color information.
PICT file type:	A data-fork file that contains a 512 byte header, followed by a QuickDraw picture.
PICT resource type:	This is a resource which contains a QuickDraw picture (version 1 or 2).
opcode:	A pre-defined number within the QuickDraw picture that the QuickDraw function DrawPicture uses to determine what object to draw or what mode to change for subsequent drawing.
graphic primitive:	A data structure that specifies the geometry of basic graphical shapes, such as lines, arcs, ellipses, and rectangles.

A standard format: the advantage to you as a developer

One of the outstanding features of the Macintosh computer has been the ease with which the user accesses graphic data created by different applications. Apple supports the PICT data file as the vehicle for importing and managing scanned bitMaps and pixMaps into Apple/third party applications. Developers interested in creating or modifying their applications to be compatible with Apples's graphic products should specify bit map or pixel map data using the QuickDraw picture structure described in this document. The graphic information can then be saved as a PICT data file. Because Apple supports this file format and picture structure, any application that recognizes the PICT file format should be able to read, display, and modify graphic data created by other Macintosh applications.

The PICT file (defined in *Macintosh Technical Note #27*) is a data fork file with a 512-byte header, followed by a picture (see Figure 1). This data fork file contains a QuickDraw (and now, Color QuickDraw) data structure within which a graphic application (using standard QuickDraw calls) places drawing primitives to represent an object or image graphic data. In the QuickDraw picture format, pictures consist of opcodes followed by picture data.

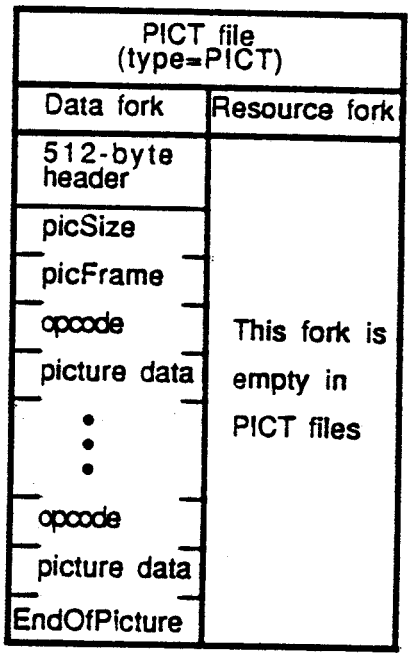


Figure 1. PICT file format.

With the introduction of Macintosh II, the QuickDraw picture structure has been extended to include new color graphics opcodes, as well as enable future expandibility. The new opcodes solve many of the major problems encountered by developers in using PICT files. For example, it is now possible to specify the resolution of bitmap data. Color can also be specified, but only chunky pixels (contiguously stored pixel components) are currently recognized by Color QuickDraw. Your application only needs to generate or recognize the chunky pixel format. This format is indicated by an image or pixMap with a cmpCount =1.

Most existing applications work fine, without modification, with version 2 pictures. On a Macintosh II, version 2 pictures will draw in color (if drawn directly to the screen). Currently, they will print using the old QuickDraw colors. Eventually, new print drivers will be able to take advantage of the new color information.

On a Macintosh 512 enhanced, Macintosh Plus, and Macintosh SE, a patch in the System file beginning with version 4.1 provides QuickDraw with the capability to convert and display version 2 pictures. The original Macintosh and Macintosh 512 cannot display version 2 pictures.

Applications that generate pictures in the QuickDraw picture format are free to use any or all available features to support their particular needs. Some will use only the pixMap primitive. You may wish to include comments in the picture that are pertinent to the needs of your application. In general, put a minimal amount of information in your PICT files and avoid redundancy. It is reasonable for receiving applications to ignore picture opcodes that are not needed.

For those developers interested in generating picture data on non-Apple CPUs, the QuickDraw internal opcodes are included at the back of this document. For example, the university community may wish to generate ray-traced images on mainframes and display them in the Macintosh environment through use of the PICT file format and QuickDraw picture structure.

Use of QuickDraw Picture Format for Image Data

For developers, there are advantages to using the QuickDraw picture format to display images in their graphics applications:

- Because the PICT structure is supported by Apple at the system level, it will always be considered in upgrade paths to new architectures.
- Establishing a standard format for image data has enormous advantages for software developers interested in graphic data interchange between applications. If, for example, all scanner peripherals support PICT as the graphic data file format, the task of importing data would then be reduced to parsing a single file format.

Key differences between version 1 & 2 pictures

The major differences between version 1 and 2 pictures are listed below.

- version 1 opcodes are a single byte; version 2 opcodes are 2 bytes in length. This means that old opcodes in a version 2 picture take up two bytes, not one.
- version 1 data may start on byte boundaries; version 2 data is always word-aligned.
- in version 2, the high bit of the rowBytes field is used to indicate pixMap instead of bitMap; pixData then replaces bitData.
- all unused version 2 opcodes, as well as the number of data bytes associated with each, have been defined. This was done so that picture parsing code can safely ignore unknown opcodes, enabling future use of these opcodes in a backward-compatible manner.

Picture parsing

The first 512 bytes of a PICT data file contain application-specific header information. Each QuickDraw (and Color QuickDraw) picture definition consists of a fixed-size header containing information about the size, scaling, and version of the picture, followed by the opcodes and picture data defining the objects drawn between the OpenPicture and ClosePicture calls.

When the OpenPicture routine is called and the port is an old grafPort, a version 1 picture is opened. When the OpenPicture routine is called and the port is a CGrafPort, then a version 2 picture is opened. If any fields in the grafPort are different than the default entries, those fields that are different get recorded in the picture.

Version 4.1 of the Macintosh System file incorporates a patch to QuickDraw that will enable QuickDraw (on machines with 128K or larger ROMs) to parse a version 2 PICT file, read it completely, attempt to convert all Color QuickDraw color opcodes to a suitable black-and-white representation, and draw the picture in an old grafPort. If you are trying to display a version 2 picture on a Macintosh without the system patch, QuickDraw won't be able to read (or display) the picture data but it shouldn't crash your machine.

On the Macintosh II, old pictures can also be played back in new grafPorts. You should only use old drawing commands in old pictures. In new pictures, old and new drawing commands can be intermixed.

How QuickDraw defines a picture

The Pascal record structure of version 1 and 2 pictures is exactly the same. In both, the picture begins with a picSize, then a picFrame (rect), followed by the picture definition data. Since a picture may include any sequence of drawing commands, its data structure is a variable-length entity. It consists of two fixed-length fields followed by a variable-length field:

Record structure of QuickDraw & Color QuickDraw pictures

```
TYPE Picture = RECORD
                  picSize:    INTEGER; {low order 16 bits of picture size;
                                        this is not useful information}
                  picFrame:   Rect;    {picture frame, used as reference for
                                        scaling when the picture is drawn}
                   {picture definition data}
               END;
```

To maintain compatibility with the original picture format, the picSize field has not been changed in version 2 pictures. However, the information in this field is only useful if your application supports version 1 pictures not exceeding 32K bytes in size. Because pictures can be much larger than the 32K limit imposed by the 2-byte picSize field, use the GetHandleSize call to determine picture size if the picture is in memory or the file size returned in pBFGetInfo if the picture resides in a file.

The picFrame field is the picture frame that surrounds the picture and gives a frame of reference for scaling when the picture is played back. The rest of the structure contains a compact representation of the image defined by the opcodes. The picture definition data consists of a sequence of the opcodes listed in Table 2, each followed by zero or more bytes of data. Every opcode has an implicit or explicit size associated with it that indicates the number of data bytes following that opcode, ranging from 2 to 2^{32} bytes (this maximum number of bytes applies to version 2 pictures only).

Picture spooling

In the past, images rarely exceeded the 32K practical limit placed on resources. Today, with the advent of scanners and other image input products, images may easily exceed this size. This increase in image size necessitates a means for handling pictures that are too large to reside entirely in memory. One solution is to place the picture in the data fork of a PICT file, and spool it in as needed. To read the file, an application need simply replace the QuickDraw default getPicProc routine with a procedure (getPICTData) that reads the picture data from a disk file; the disk access would be transparent. Note that this technique applies equally to version 1 (byte-opcode) and version 2 (word-opcode) pictures.

Spooling a picture from disk

In order to display pictures of arbitrary size, an application must be able to import a QuickDraw picture from a file of type PICT. (This is the file type produced by a Save as... from MacDraw with the PICT option selected.) What follows is a small program fragment that demonstrates how to spool in a picture from the data fork of a PICT file. The picture can be larger than the historical 32K resource size limitation.

> **Note:** *To better understand how to provide additional error checking during picture spooling, refer to Macintosh Technical Notes #154, "Displaying Large PICT Files".*

```
{ the following variable and procedure must be at the main level of the program }
  VAR
      globalRef: INTEGER;

  PROCEDURE GetPICTData(dataPtr: Ptr; byteCount: INTEGER); { replacement for
                                                             getPicProc routine
      VAR
          err : INTEGER;
          longCount: LONGINT;

      BEGIN
          longCount := byteCount; { longCount is a Pascal VAR parameter and
                                    must be a LONGINT }
          err := FSRead(globalRef,longCount,dataPtr);
          { ignore errors here since it is unclear how to handle them }
      END;
```

```
PROCEDURE GetandDrawPICTFile; { procedure to draw in a picture from a PICT file
                                selected by the user }

    VAR
        wher: Point; { where to display dialog }
        reply: SFReply; { reply record }
        myFileTypes: SFTypeList; { more of the Standard File goodies }
        NumFileTypes: INTEGER;
        err: OSErr;
        myProcs: QDProcs; { use CQDProcs for a CGrafPort (a color window) }
        PICTHand: PicHandle; { we need a picture handle for DrawPicture }
        longCount: LONGINT;

        myPB: ParamBlockRec;

    BEGIN
        wher.h := 20;
        wher.v := 20;
```

```
            NumFileTypes := 1; { Display PICT files }
            myFileTypes[0] := 'PICT';
            SFGetFile(wher,'',NIL,NumFileTypes,myFileTypes,NIL,reply);
            IF reply.good THEN BEGIN
                err := FSOpen(reply.fname,reply.vrefnum,globalRef);

                SetStdProcs(myProcs); { use SetStdCProcs for a CGrafPort }
                myWindow^.grafProcs := @myProcs;
                myProcs.getPicProc := @GetPICTData;

                PICTHand := PicHandle(NewHandle(SizeOf(Picture))); { get one
                            the size of (size word + frame rectangle) }

                { skip (so to speak) the MacDraw header block }
                err := SetFPos(globalRef,fsFromStart,512);
                longCount := SizeOf(Picture);
                { read in the (obsolete) size word and the picture frame }
                err := FSRead(globalRef,longCount,Ptr(PICTHand^));

                DrawPicture(PICTHand,PICTHand^^.picFrame);
                { inside of DrawPicture, QD makes repeated calls to getPicProc
                  to get actual picture opcodes and data. Since we have
                  intercepted GetPicProc, QD will call myProcs to get
                  getPicProc, instead of calling the default procedure }

                err := FSClose(globalRef);

                myWindow^.grafProcs := NIL;
                DisposHandle(Handle(PICTHand));

            END; { IF reply.good }
        END;
```

Spooling a picture to a file

Spooling a picture out to a file is equally straightforward. By replacing the standard putPicProc with your own procedure, you can create a PICT file and spool the picture data out to the file.

Drawing to an offscreen pixel map to get interactive performance

With the advent of high resolution output devices such as laser printers, the need has arisen to support bit-map images at resolutions higher than those supported by the screen. In order to speed up the interactive manipulation of high-resolution pixel-map images, developers may want to first draw them into an off-screen pixel map at screen resolution and retain this screen version as long as the document is open.

> **Note:** *You can use the formula shown under the section on "Sample PICT file" to calculate the resolution of the source data. How to draw into an offscreen pixMap is described in Macintosh Technical Note #120.*

A new set of grafProcs routines

The entire opcode space has been defined or reserved, as shown in Table 2, and a new set of routines has been added to the grafProcs record. The purpose of these changes is to provide support for anticipated future enhancements in a way that will not cause old applications to crash. How this works is that when Color QuickDraw encounters an unused opcode, it calls the new opcodeProc routine to parse the opcode data. By default, this routine simply ignores the data, since no new opcodes are defined (other than HeaderOp, which is also ignored).

Color QuickDraw has replaced the QDProcs record with a CQDProcs record. In a new grafPort, you should never use the SetStdProcs routine. If you do, it will return the old QDProcs record, which will not contain an entry for the stdOpcodeProc. If you do not use the new SetStdCProcs routine to parse the unused opcodes, the first color picture that you try to display may crash your system.

Extensions to the QDPROCS record

```
;
opcodeProc      EQU  $34      [pointer]
newProc1        EQU  $38      [pointer]
newProc2        EQU  $3C      [pointer]
newProc3        EQU  $40      [pointer]
newProc4        EQU  $44      [pointer]
newProc5        EQU  $48      [pointer]
newProc6        EQU  $4C      [pointer]
CQDProcsRec     EQU  $50      size of QDProcs record
```

An Eye Toward Compatibility

Many applications already support PICT resources larger than 32K. The 128K ROMs (and later) allow pictures as large as memory (or spooling) will accommodate. This was made possible by having QuickDraw ignore the size word and simply read the picture until the end-of-picture opcode is reached.

For maximum safety and convenience, let QuickDraw generate and interpret your pictures.

While Apple has provided you with the data formats that allow you to read or write picture data directly, we recommend that you always let DrawPicture or OpenPicture and ClosePicture process the opcodes.

One reason to read a picture directly by scanning the opcodes would be to disassemble it to, for example, extract a Color QuickDraw pixel map to save off in a private data structure. This shouldn't normally be necessary, unless you are working with an application running on a CPU other than the Macintosh. You wouldn't need to do it, of course, if you were using Color QuickDraw.

If you do look at the picture data, be sure and check the version information. You may want to include an alert (dialog box) in your application that indicates to the user when a picture was created using a later version of the picture format than currently recognized by your application, letting them know that some elements of the picture cannot be displayed. If the version information indicates a QuickDraw picture version later than the one recognized by your application, your program should skip over the new opcodes and only attempt to parse the opcodes it knows.

As with reading picture data directly, it is best to use QuickDraw to create data in the PICT format. If you do have a need to create PICT format data directly, it is essential that you understand and follow the format presented in Table 2 and thoroughly test the data produced on both color and black and white Macintosh machines.

Apple does not guarantee that a picture which wasn't produced by QuickDraw will work.

Picture Format

This section describes the internal structure of the QuickDraw picture, consisting of a fixed-length header (which is different for version 1 and version 2 pictures), followed by variable-sized picture data. Your picture structure must follow the order shown in the examples below.

The two fixed-length fields, picSize and picFrame, are the same for version 1 and version 2 pictures.

```
picSize:    INTEGER;  {low-order 16 bits of picture size}
picFrame:   RECT;     {picture frame, used as scaling reference}
```

Following these fields is a variable amount of opcode-driven data. Opcodes represent drawing commands and parameters that affect those drawing commands in the picture. The first opcode in any picture must be the version opcode, followed by the version number of the picture.

Picture Definition: Version 1

Picture Header - fixed size of 2 bytes:

```
$11          BYTE     {version opcode}
$01          BYTE     {version number of picture}
```

Picture Definition Data - variable sized:

```
opcode       BYTE     {one drawing command}
data . . .
opcode       BYTE     {one drawing command}
data . . .

$FF                   {end-of-picture opcode}
```

In a version 1 picture, the version opcode is $11, which is followed by version number $01. When parsing a version 1 picture, Color QuickDraw (or a patched QuickDraw) assumes it is reading an old picture, fetching a byte at a time as opcodes. An end-of-picture byte ($FF) after the last opcode or data byte in the file signals the end of the data stream.

Picture Definition: Version 2

Picture Header - fixed size of 30 bytes:

```
$0011        WORD     {version opcode}
$02FF        WORD     {version number of new picture}

$0C00        WORD     {reserved header opcode}
24 bytes of data      {reserved for future Apple use}
```

Picture Definition Data - variable sized:

```
opcode       WORD     {one drawing command}
data . . .
opcode       WORD     {one drawing command}
data . . .

$00FF        WORD     {end-of-picture opcode}
```

In a version 2 picture, the first opcode is a two-byte version opcode ($0011). This is followed by a two-byte version number ($02FF). On machines without the 4.1 System File, the first $00 byte is interpreted as a no-op and is skipped, then the $11 is interpreted as a version opcode. On a Macintosh II (or a Macintosh with System File 4.1 or later), this identifies the picture as a version 2 picture, and all subsequent opcodes are read as words (which are word-aligned within the picture). On a Macintosh without the 4.1 System patch, the $02 is read as the version number, then the $FF is read and interpreted as the end-of-picture opcode. For this reason, DrawPicture terminates without drawing anything.

For future expandibility, the second opcode in every version 2 picture must be a reserved header opcode, followed by 24 bytes of data that are not used by your application.

picComments

If your application requires capability beyond that provided by the picture opcodes, the picComment opcode allows data or commands to be passed directly to the output device. picComments enable MacDraw, for example, to reconstruct graphics primitives not found in QuickDraw (e.g., rotated text) that are received either from the clipboard or from another application. picComments are also used as a means of communicating more effectively with the LaserWriter and with other applications via the scrap or the PICT data file.

Because some operations (like splines and rotated text) can be implemented more efficiently by the LaserWriter, some of the picture comments are designed to be issued along with QuickDraw commands that simulate the commented commands on the Macintosh screen. If the printer you are printing to has not implemented the comment commands, it ignores them and simulates the operations using the accompanying QuickDraw commands. Otherwise, it uses the comments to implement the desired effect and ignores the appropriate QuickDraw-simulated commands.

> **Note:** *The picture comments used by MacDraw are listed and described in Macintosh Technical Note #27.*

If you are going to be producing or modifying your own picture, the structure and use of these comments must be precise. The comments and the embedded QuickDraw commands must come in the correct sequence in order to work properly.

> **Note:** *Apple is currently investigating a method to register picComments. If you intend to use new picComments in your application, you must contact Apple's Developer Technical Support to avoid conflict with picComment numbers used by other developers.*

Sample PICT file

An example of a version 2 picture data file which can display a single image is shown on the following page. Applications that generate picture data should set the resolution of the image source data in the hRes and vRes fields of the PICT file. We recommend, however, that you calculate the image resolution anyway using the values for srcRect and dstRect according to the following formulas:

$$\text{horizontal resolution (hRes)} = \frac{\text{width of srcRect}}{\text{width of dstRect}} \times 72$$

$$\text{vertical resolution (vRes)} = \frac{\text{height of srcRect}}{\text{height of dstRect}} \times 72$$

PICT file example

Size (in bytes)	Name	Description
2	picSize	low word of picture size
8	picFrame	rectangular bounding box of picture, at 72 dpi
Picture Definition Data:		
2	version op	version opcode = $0011
2	version	version number = $02FF
2	Header op	header opcode = $0C00
4	size	total size of picture in bytes (=-1 for v.2 pictures)
16	fBBox	fixed-point bounding box (=-1 for v.2 pictures)
4	reserved	reserved for future Apple use (=-1 for v.2 pict.)
2	opbitsRect	BitMap opcode = $0090
2	rowBytes	integer, must have high bit set to signal pixMap
8	bounds	rectangle, bounding rectangle at source resolution
2	pmVersion	integer, pixMap version number
2	packType	integer, defines packing format
4	packsize	LongInt, length of pixel data
4	hRes	fixed, horizontal resolution (dpi) of source data
4	vRes	fixed, vertical resolution (dpi) of source data
2	pixelType	integer, defines pixel type
2	pixelSize	integer, number of bits in pixel
2	cmpCount	integer, number of components in pixel
2	cmpSize	integer, number of bits per component
4	planeBytes	LongInt, offset to next plane
	pmTable	color table = 0
	pmReserved	reserved = 0
4	ctseed	LongInt, color table seed
2	transIndex	integer, index of transparent pixels
2	ctSize	integer, number of entries in CTTable
(ctSize+1) * 8	CTTable	color lookup table data
8	srcRect	rectangle, source rectangle at source resolution
8	dstRect	rectangle, destination rectangle at 72 dpi resolution
2	mode	integer, transfer mode
see Table 4	pixData	pixel data
2	endPICT op	end-of-picture opcode = $00FF

PICT Opcodes

The opcode information in Table 2 (on pages 15-19) is provided for the purpose of debugging application-generated PICT files. Your application should generate and read PICT files only by using standard QuickDraw or Color QuickDraw routines (OpenPicture, ClosePicture).

The data types listed below are used in the in Table 2 opcode definitions. Data formats are described in *Inside Macintosh, Volume I*.

Table 1. Data types

Type	Size
v1 opcode	1 byte
v2 opcode	2 bytes
integer	2 bytes
long integer	4 bytes
mode	2 bytes
point	4 bytes
0..255	1 byte
–128..127	1 byte (signed)
rect	8 bytes (top, left, bottom, right: integer)
poly	10+ bytes
region	10+ bytes
fixed-point number	4 bytes
pattern	8 bytes
rowbytes	2 bytes (always an even quantity)

Valid picture opcodes are listed in Table 2. New opcodes or those altered for version 2 picture files are indicated by a leading asterisk (*). Refer to *Inside Macintosh, volume V* for specific details on the new Color QuickDraw routines. The unused opcodes found throughout the table are reserved for Apple use. The length of the data that follows these opcodes is pre-defined, so if they are encountered in pictures, they can simply be skipped. By default, Color QuickDraw reads and then ignores these opcodes.

Notes For Table 2:

1. *The opcode value has been extended to a word for version 2 pictures. Remember, opcode size = 1 byte for version 1.*

2. *Because opcodes must be word aligned in version 2 pictures, a byte of 0 (zero) data is added after odd-size data.*

3. *The size of reserved opcodes has been defined. They can occur only in version 2 pictures.*

4. *All unused opcodes are reserved for future Apple use and should not be used.*

5. *For opcodes $0040 - $0044: rounded-corner rectangles use the setting of the ovSize point (refer to opcode $000B)*

6. *For opcodes $0090 and $0091: data is unpacked. These opcodes can only be used for rowbytes less than 8.*

7. *For opcodes $0100 - $7FFF: the amount of data for opcode $nnXX = 2 * nn bytes*

Table 2. PICT opcodes

Opcode	Name	Description	Data Size (in bytes)
$0000	NOP	nop	0
$0001	Clip	clip	region size
$0002	BkPat	background pattern	8
$0003	TxFont	text font (word)	2
$0004	TxFace	text face (byte)	1
$0005	TxMode	text mode (word)	2
$0006	SpExtra	space extra (fixed point)	4
$0007	PnSize	pen size (point)	4
$0008	PnMode	pen mode (word)	2
$0009	PnPat	pen pattern	8
$000A	FillPat	fill pattern	8
$000B	OvSize	oval size (point)	4
$000C	Origin	dh, dv (word)	4
$000D	TxSize	text size (word)	2
$000E	FgColor	foreground color (long)	4
$000F	BkColor	background color (long)	4
$0010	TxRatio	numer (point), denom (point)	8
$0011	Version	version (byte)	1
$0012	*BkPixPat	color background pattern	variable: see Table 3
$0013	*PnPixPat	color pen pattern	variable: see Table 3
$0014	*FillPixPat	color fill pattern	variable: see Table 3
$0015	*PnLocHFrac	fractional pen position	2
$0016	*ChExtra	extra for each character	2
$0017	*reserved for Apple use	opcode	0
$0018	*reserved for Apple use	opcode	0
$0019	*reserved for Apple use	opcode	0
$001A	*RGBFgCol	RGB foreColor	variable: see Table 3
$001B	*RGBBkCol	RGB backColor	variable: see Table 3
$001C	*HiliteMode	hilite mode flag	0
$001D	*HiliteColor	RGB hilite color	variable: see Table 3
$001E	*DefHilite	Use default hilite color	0
$001F	*OpColor	RGB OpColor for arithmetic modes	variable: see Table 3
$0020	Line	pnLoc (point), newPt (point)	8
$0021	LineFrom	newPt (point)	4
$0022	ShortLine	pnLoc (point, dh, dv (-128. . 127)	6
$0023	ShortLineFrom	dh, dv (-128. . 127)	2
$0024	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$0025	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$0026	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$0027	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$0028	LongText	txLoc (point), count (0. .255), text	5 + text
$0029	DHText	dh (0. .255), count (0. .255), text	2 + text
$002A	DVText	dv (0. .255), count (0. .255), text	2 + text
$002B	DHDVText	dh, dv (0. .255), count (0. .255), text	3 + text
$002C	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$002D	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$002E	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length

Table 2. PICT opcodes *(continued)*

Opcode	Name	Description	Data Size (in bytes)
$002F	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$0030	frameRect	rect	8
$0031	paintRect	rect	8
$0032	eraseRect	rect	8
$0033	invertRect	rect	8
$0034	fillRect	rect	8
$0035	*reserved for Apple use	opcode + 8 bytes data	8
$0036	*reserved for Apple use	opcode + 8 bytes data	8
$0037	*reserved for Apple use	opcode + 8 bytes data	8
$0038	frameSameRect	rect	0
$0039	paintSameRect	rect	0
$003A	eraseSameRect	rect	0
$003B	invertSameRect	rect	0
$003C	fillSameRect	rectangle	0
$003D	*reserved for Apple use	opcode	0
$003E	*reserved for Apple use	opcode	0
$003F	*reserved for Apple use	opcode	0
$0040	frameRRect	rect (see *Note # 5* on page 13)	8
$0041	paintRRect	rect (see *Note # 5* on page 13)	8
$0042	eraseRRect	rect (see *Note # 5* on page 13)	8
$0043	invertRRect	rect (see *Note # 5* on page 13)	8
$0044	fillRRect	rect (see *Note # 5* on page 13)	8
$0045	*reserved for Apple use	opcode + 8 bytes data	8
$0046	*reserved for Apple use	opcode + 8 bytes data	8
$0047	*reserved for Apple use	opcode + 8 bytes data	8
$0048	frameSameRRect	rect	0
$0049	paintSameRRect	rect	0
$004A	eraseSameRRect	rect	0
$004B	invertSameRRect	rect	0
$004C	fillSameRRect	rect	0
$004D	*reserved for Apple use	opcode	0
$004E	*reserved for Apple use	opcode	0
$004F	*reserved for Apple use	opcode	0
$0050	frameOval	rect	8
$0051	paintOval	rect	8
$0052	eraseOval	rect	8
$0053	invertOval	rect	8
$0054	fillOval	rect	8
$0055	*reserved for Apple use	opcode + 8 bytes data	8
$0056	*reserved for Apple use	opcode + 8 bytes data	8
$0057	*reserved for Apple use	opcode + 8 bytes data	8
$0058	frameSameOval	rect	0
$0059	paintSameOval	rect	0
$005A	eraseSameOval	rect	0
$005B	invertSameOval	rect	0
$005C	fillSameOval	rect	0

Table 2. PICT opcodes *(continued)*

Opcode	Name	Description	Data Size (in bytes)
$005D	*reserved for Apple use	opcode	0
$005E	*reserved for Apple use	opcode	0
$005F	*reserved for Apple use	opcode	0
$0060	frameArc	rect, startAngle, arcAngle	12
$0061	paintArc	rect, startAngle, arcAngle	12
$0062	eraseArc	rect, startAngle, arcAngle	12
$0063	invertArc	rect, startAngle, arcAngle	12
$0064	fillArc	rect, startAngle, arcAngle	12
$0065	*reserved for Apple use	opcode + 12 bytes	12
$0066	*reserved for Apple use	opcode + 12 bytes	12
$0067	*reserved for Apple use	opcode + 12 bytes	12
$0068	frameSameArc	rect	4
$0069	paintSameArc	rect	4
$006B	invertSameArc	rect	4
$006C	fillSameArc	rect	4
$006D	*reserved for Apple use	opcode + 4 bytes	4
$006E	*reserved for Apple use	opcode + 4 bytes	4
$006F	*reserved for Apple use	opcode + 4 bytes	4
$0070	framePoly	poly	polygon size
$0071	paintPoly	poly	polygon size
$0072	erasePoly	poly	polygon size
$0073	invertPoly	poly	polygon size
$0074	fillPoly	poly	polygon size
$0075	*reserved for Apple use	opcode + poly	
$0076	*reserved for Apple use	opcode + poly	
$0077	*reserved for Apple use	opcode word + poly	
$0078	frameSamePoly	(not yet implemented - same as 70, etc.)	0
$0079	paintSamePoly	(not yet implemented)	0
$007A	eraseSamePoly	(not yet implemented)	0
$007B	invertSamePoly	(not yet implemented)	0
$007C	fillSamePoly	(not yet implemented)	0
$007D	*reserved for Apple use	opcode	0
$007E	*reserved for Apple use	opcode	0
$007F	*reserved for Apple use	opcode	0
$0080	frameRgn	rgn	region size
$0081	paintRgn	rgn	region size
$0082	eraseRgn	rgn	region size
$0083	invertRgn	rgn	region size
$0084	fillRgn	rgn	region size
$0085	*reserved for Apple use	opcode + rgn	region size
$0086	*reserved for Apple use	opcode + rgn	region size
$0087	*reserved for Apple use	opcode + rgn	region size
$0088	frameSameRgn	(not yet implemented - same as 80, etc.)	0
$0089	paintSameRgn	(not yet implemented)	0
$008A	eraseSameRgn	(not yet implemented)	0

Table 2. PICT opcodes *(continued)*

Opcode	Name	Description	Data Size (in bytes)
$008B	invertSameRgn	(not yet implemented)	0
$008C	fillSameRgn	(not yet implemented)	0
$008D	*reserved for Apple use	opcode	0
$008E	*reserved for Apple use	opcode	0
$008F	*reserved for Apple use	opcode	0
$0090	*BitsRect	copybits, rect clipped	variable: see Table 3
$0091	*BitsRgn	copybits, rgn clipped	variable: see Table 3
$0092	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$0093	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$0094	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$0095	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$0096	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$0097	*reserved for Apple use	opcode word + 2 bytes data length + data	2+ data length
$0098	*PackBitsRect	packed copybits, rect clipped	variable: see Table 3
$0099	*PackBitsRgn	packed copybits, rgn clipped	variable: see Table 3
$009A	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$009B	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$009C	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$009D	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$009E	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$009F	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$00A0	ShortComment	kind (word)	2
$00A1	LongComment	kind (word), size (word), data	4+data
$00A2	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
:	:	:	
$00AF	*reserved for Apple use	opcode + 2 bytes data length + data	2+ data length
$00B0	*reserved for Apple use	opcode	0
:	:	:	
$00CF	*reserved for Apple use	opcode	0
$00D0	*reserved for Apple use	opcode + 4 bytes data length + data	4+ data length
:	:	:	
$00FE	*reserved for Apple use	opcode + 4 bytes data length + data	4+ data length
$00FF	opEndPic	end of picture	2
$0100	*reserved for Apple use	opcode + 2 bytes data	2
:	:	:	
$01FF	*reserved for Apple use	opcode + 2 bytes data	2

Table 2. PICT opcodes *(continued)*

Opcode	Name	Description	Data Size (in bytes)
$0200	*reserved for Apple use	opcode + 4 bytes data	4
:	:	:	
$0BFF	*reserved for Apple use	opcode + 4 bytes data	22
$0C00	HeaderOp	opcode	24
$0C01:	*reserved for Apple use	opcode + 4 bytes data	24
:	:	:	
$7F00	*reserved for Apple use	opcode + 254 bytes data	254
:	:	:	
$7FFF	*reserved for Apple use	opcode + 254 bytes data	254
$8000	*reserved for Apple use	opcode	0
:	:	:	
$80FF	*reserved for Apple use	opcode	0
$8100	*reserved for Apple use	opcode + 4 bytes data length + data	4+ data length
:	:	:	
$FFFF	*reserved for Apple use	opcode + 4 bytes data length + data	4+ data length

The new opcodes - expanded format

The expanded format of the version 2 PICT opcodes are shown in Table 3 below.

Table 3. Data format of version 2 PICT opcodes

Opcode	Name	Description
$0012	**BkPixPat**	**color background pattern**
$0013	**PnPixPat**	**color pen pattern**
$0014	**FillPixPat**	**color fill pattern**

```
if patType = ditherPat
  then
        PatType: word;        { pattern type = 2 }
        Pat1Data: Pattern;    { old pattern data }
        RGB: RGBColor;        { desired RGB for pattern }
  else
        PatType: word;        { pattern type = 1 }
        Pat1Data: Pattern;    { old pattern data }
        pixMap:               { pixMap format shown below }
        colorTable:           { color table format shown below }

        pixData:              { pixData format shown below
  end;
```

Table 3. Data format of version 2 PICT opcodes *(continued)*

Opcode	Name	Description
$0015	**PnLocHFrac**	**fractional pen position**

```
pnLocHFrac: word;        { see Inside Macintosh for format }
```

If `pnLocHFrac` <> 1/2, it is always put to the picture before each text drawing operation.

$0016	**ChExtra**	**extra for each character**

```
chExtra: word;           { see Inside Macintosh for format }
```

After chExtra changes, it is put to picture before next text drawing operation.

$001A	**RGBFgCol**	**RGB foreColor**
$001B	**RGBBkCol**	**RGB backColor**
$001D	**HiliteColor**	**RGB hilite color**
$001F	**OpColor**	**RGB OpColor for arithmetic modes**

```
RGB: RGBColor;           { desired RGB for fg/bk }
```
(see *Inside Macintosh, Volume V* for data structure)

$001C	**HiliteMode**	**hilite mode flag**

No data. This opcode is sent before a drawing operation that uses the hilite mode.

$001E	**DefHilite**	**Use default hilite color**

No data. Set hilite to default (from low memory).

The next four opcodes ($0090, $0091, $0098, $0099) are modifications of existing (version 1) opcodes. The first word following the opcode is the rowBytes. If the high bit of the rowBytes is set, then it is a pixMap containing multiple bits per pixel; if it is not set, it is a bitMap containing one bit per pixel. In general, the difference between version 1 and 2 formats is that the pixMap replaces the bitMap, a color table has been added, and pixData replaces the bitData.

> **Note:** *Opcodes $0090 and $0091 are only used for rowbytes less than 8.*

$0090	**BitsRect**	**copybits, rect clipped**

```
pixMap:                  { described in Table 4 }
colorTable:              { described in Table 4 }
srcRect:      Rect;      { source rectangle }
dstRect:      Rect;      { destination rectangle }
mode:         Word;      { transfer mode (may include new transfer modes) }
PixData:                 { described in Table 4 }
```

Table 3. Data format of version 2 PICT opcodes *(continued)*

Opcode	Name	Description
$0091	**BitsRgn**	**copybits, rgn clipped**
	pixMap:	{ described in Table 4 }
	colorTable:	{ described in Table 4 }
	srcRect: Rect;	{ source rectangle }
	dstRect: Rect;	{ destination rectangle }
	mode: Word;	{ transfer mode (may include new transfer modes) }
	maskRgn: Rgn;	{ region for masking }
	PixData:	{ described in Table 4 }
$0098	**PackBitsRect**	**packed copybits, rect clipped**
	pixMap:	{ described in Table 4 }
	colorTable:	{ described in Table 4 }
	srcRect: Rect;	{ source rectangle }
	dstRect: Rect;	{ destination rectangle }
	mode: Word;	{ transfer mode (may include new transfer modes) }
	PixData:	{ described in Table 4 }
$0099	**PackBitsRgn**	**packed copybits, rgn clipped**
	pixMap:	{ described in Table 4 }
	colorTable:	{ described in Table 4 }
	srcRect: Rect;	{ source rectangle }
	dstRect: Rect;	{ destination rectangle }
	mode: Word;	{ transfer mode (may include new transfer modes) }
	maskRgn: Rgn;	{ region for masking }
	PixData:	{ described in Table 4 }

Table 4. Data types found within new PICT opcodes listed in Table 3

Opcode	Name	Description
pixMap =	baseAddr: long;	{ unused = 0 }
	rowBytes: word;	{ rowBytes w/high byte set }
	Bounds: rect;	{ bounding rectangle }
	version: word;	{ version number = 0 }
	packType: word;	{ packing format = 0 }
	packSize: long;	{ packed size = 0 }
	hRes: fixed;	{ horizontal resolution (default = $0048.0000) }
	vRes: fixed;	{ vertical resolution (default = $0048.0000) }
	pixelType: word;	{ chunky format = 0 }
	pixelSize: word;	{ # bits per pixel (1,2,4,8) }
	cmpCount: word;	{ # components in pixel = 1 }
	cmpSize: word;	{ size of each component = pixelSize for chunky }
	planeBytes: long;	{ offset to next plane = 0 }
	pmTable: long;	{ color table = 0 }
	pmReserved: long;	{ reserved = 0 }
end;		

Table 4. Data types found within new PICT opcodes listed in Table 3 *(continued)*

Opcode	Name	Description

```
colorTable = ctseed:      long;     { id number for color table = 0 }
             transIndex: word;      { flags word = 0 }
             ctSize:      word;     { number of ctTable entries-1 }
                                    { ctSize + 1 color table entries }
                                    { each entry = pixel value, red, green, blue: word }
  end;

pixData:     If rowBytes < 8 then data is unpacked
                  data size = rowBytes*(bounds.bottom-bounds.top);
             If rowBytes >= 8 then data is packed.
                  Image contains (bounds.bottom-bounds.top) packed scanlines.
                  Packed scanlines are produced by the PackBits routine.
                  Each scanline consists of [byteCount] [data].
                  If rowBytes > 250 then byteCount is a word, else it is a byte.
  end;
```