New Technical Notes

Macintosh

Developer Support

QuickDraw's Internal Picture Definition

Imaging

Revised by: Rick Blair November 1986
March 1988
Written by: Ginger Jenigan April 1985

This technical note describes the internal format of the QuickDraw picture data structure. This revision corrects some errors in the opcode descriptions and provides some examples.

This technical note describes the internal definition of the QuickDraw picture. The information given here only applies to QuickDraw picture format version 1.0 (which is always created by Macintoshes without Color QuickDraw). Picture format version 2.0 is documented in the Color QuickDraw chapter of *Inside Macintosh*. This information should not be used to write your own picture bottleneck procedures; if we add new objects to the picture definition, your program will not be able to operate on pictures created using standard QuickDraw. Your program will not know the size of the new objects and will, therefore, not be able to proceed past the new objects. (What this ultimately means is that you can't process a new picture with an old bottleneck proc.)

Terms

An *opcode* is a number that `DrawPicture` uses to determine what object to draw or what *mode* to change for subsequent drawing. The following list gives the opcode, the name of the object (or mode), the associated data, and the total size of the opcode and data. To better interpret the sizes, please refer to page I-91 of the Using Assembly Language chapter of *Inside Macintosh*. For types not described there, here is a quick list:

opcode	byte
mode	word
point	4 bytes
0..255	byte
–128..127	signed byte
rect	8 bytes
poly	10+ bytes (starts with word size for poly (incl. size word)
region	10+ bytes (starts with word size for region (incl. size word)
fixed point	long

pattern	8 bytes
rowbytes	word (always even)
bit data	rowbytes * (bounds.bottom - bounds.top) bytes

Each picture definition begins with a `picsize` (`word`), then a `picframe` (`rect`), and then the picture definition, which consists of a combination of the following opcodes:

Opcode	Name	Additional Data	Total Size (bytes)
00	NOP	none	1
01	clipRgn	rgn	1+rgn
02	bkPat	pattern	9
03	txFont	font (word)	3
04	txFace	face (byte)	2
05	txMode	mode (word)	3
06	spExtra	extra (fixed point)	5
07	pnSize	pnSize (point)	5
08	pnMode	mode (word)	3
09	pnPat	pattern	9
0A	thePat	pattern	9
0B	ovSize	point	5
0C	origin	dh, dv (words)	5
0D	txSize	size (word)	3
0E	fgColor	color (long)	5
0F	bkColor	color (long)	5
10	txRatio	numer (point), denom (point)	9
11	picVersion	version (byte)	2
20	line	pnLoc (point), newPt (point)	9
21	line from	newPt (point)	5
22	short line	pnLoc (point); dh, dv (-128..127)	7
23	short line from	dh, dv (-128..127)	3
28	long text	txLoc (point), count (0..255), text	6+text
29	DH text	dh (0..255), count (0..255), text	3+text
2A	DV text	dv (0..255), count (0..255), text	3+text
2B	DHDV text	dh, dv (0..255), count (0..255), text	4+text
30	frameRect	rect	9
31	paintRect	rect	9
32	eraseRect	rect	9
33	invertRect	rect	9
34	fillRect	rect	9
38	frameSameRect	rect	1
39	paintSameRect	rect	1
3A	eraseSameRect	rect	1
3B	invertSameRect	rect	1
3C	fillSameRect	rect	1
40	frameRRect	rect (ovalwidth, height; see 1, below)	9
41	paintRRect	rect (ovalwidth, height; see 1, below)	9
42	eraseRRect	rect (ovalwidth, height; see 1, below)	9
43	invertRRect	rect (ovalwidth, height; see 1, below)	9
44	fillRRect	rect (ovalwidth, height; see 1, below)	9
48	frameSameRRect	rect	1
49	paintSameRRect	rect	1

Opcode (cont.)	Name	Additional Data	Total Size (bytes)
4A	eraseSameRRect	rect	1
4B	invertSameRRect	rect	1
4C	fillSameRRect	rect	1
50	frameOval	rect	9
51	paintOval	rect	9
52	eraseOval	rect	9
53	invertOval	rect	9
54	fillOval	rect	9
58	frameSameOval	rect	1
59	paintSameOval	rect	1
5A	eraseSameOval	rect	1
5B	invertSameOval	rect	1
5C	fillSameOval	rect	1
60	frameArc	rect, startAngle, arcAngle	13
61	paintArc	rect, startAngle, arcAngle	13
62	eraseArc	rect, startAngle, arcAngle	13
63	invertArc	rect, startAngle, arcAngle	13
64	fillArc	rect, startAngle, arcAngle	13
68	frameSameArc	startAngle, arcAngle	5
69	paintSameArc	startAngle, arcAngle	5
6A	eraseSameArc	startAngle, arcAngle	5
6B	invertSameArc	startAngle, arcAngle	5
6C	fillSameArc	startAngle, arcAngle	5
70	framePoly	poly	1+poly
71	paintPoly	poly	1+poly
72	erasePoly	poly	1+poly
73	invertPoly	poly	1+poly
74	fillPoly	poly	1+poly
78	frameSamePoly	(not yet implemented—same as 70, etc.)	1
79	paintSamePoly	(not yet implemented)	1
7A	eraseSamePoly	(not yet implemented)	1
7B	invertSamePoly	(not yet implemented)	1
7C	fillSamePoly	(not yet implemented)	1
80	frameRgn	rgn	1+rgn
81	paintRgn	rgn	1+rgn
82	eraseRgn	rgn	1+rgn
83	invertRgn	rgn	1+rgn
84	fillRgn	rgn	1+rgn
88	frameSameRgn	(not yet implemented—same as 80, etc.)	1
89	paintSameRgn	(not yet implemented)	1
8A	eraseSameRgn	(not yet implemented)	1
8B	invertSameRgn	(not yet implemented)	1
8C	fillSameRgn	(not yet implemented)	1

Opcode (cont.)	Name	Additional Data	Total Size (bytes)
90	BitsRect	rowBytes, bounds, srcRect, dstRect, mode, unpacked bitData	29+unpacked bitData
91	BitsRgn	rowBytes, bounds, srcRect, dstRect, mode, maskRgn, unpacked bitData	29+rgn+ bitData
98	PackBitsRect	rowBytes, bounds, srcRect, dstRect, mode, packed bitData for each row	29+packed bitData
99	PackBitsRgn	rowBytes, bounds, srcRect, dstRect, mode, maskRgn, packed bitData for each row	29+rgn+ packed bitData
A0	shortComment	kind(word)	3
A1	longComment	kind(word), size(word), data	5+data
FF	EndOfPicture	none	1

Notes

Rounded-corner rectangles use the setting of the ovSize point (see opcode $0B, above).

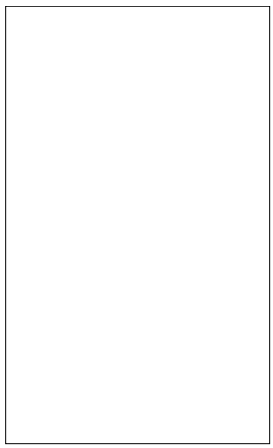
`OpenPicture` and `DrawPicture` set up a default set of port characteristics when they start. When drawing occurs, if the user's settings don't match the defaults, mode opcodes are generated. This is why there is usually a `clipRgn` code after the `picVersion`: the default clip region is an empty rectangle.

The only savings that the "same" opcodes achieve under the current implementation is for rectangles. `DrawPicture` keeps track of the last rectangle used and if a "same" opcode is encountered that requests a rectangle, the last rect. will be used (and no rectangle will appear in the opcode's data).

This last section contains some Pascal program fragments that generate pictures. Each section starts out with the picture itself (yes, they're dull) followed by the code to create and draw it, and concludes with a commented hex dump of the picture.

```
{variables used in all examples}

VAR
    err:     OSErr;
    ph:      PicHandle;
    h:       Handle;
    r:       Rect;
    smallr:  Rect;
    orgr:    Rect;
    pstate:  PenState; {are they in the Rose Bowl, or the state pen?}
```



```
I.     {Rounded-corner rectangle}
       SetRect(r, 20, 10, 120, 175);
       ClipRect(myWindow^.portRect);
       ph := OpenPicture(r);
       FrameRoundRect (r, 5, 4); {r,width,height}
       ClosePicture;
       DrawPicture(ph, r);

'PICT' (1)  0026 {size}  000A 0014 00AF 0078 {picFrame}
       1101 {version 1}  01 000A 0000 0000 00FA 0190 {clipRgn — 10 byte region}
       0B 0004 0005 {ovSize point}  40 000A 0014 00AF 0078 {frameRRect
       rectangle}
       FF {fin}
```

```
II.    {Overpainted arc}
       GetPenState(pstate); {save}
       SetRect(r, 20, 10, 120, 175);
       ClipRect(myWindow^.portRect);
       ph := OpenPicture(r);
       PaintArc(r, 3, 45); {r,startangle,endangle}
       PenPat(gray);
       PenMode(patXor); {turn the black to gray}
       PaintArc(r, 3, 45); {r,startangle,endangle}
       ClosePicture;
       SetPenState(pstate); {restore}
       DrawPicture(ph, r);

data 'PICT' (2)  0036 {size}  000A 0014 00AF 0078 {picFrame}
       1101 {version 1}  01 000A 0000 0000 00FA 0190 {clipRgn — 10 byte region}
       61 000A 0014 00AF 0078 0003 002D {paintArc rectangle,startangle,endangle}
       08 000A {pnMode patXor — note that the pnMode comes before the pnPat}
       09 AA55 AA55 AA55 AA55 {pnPat gray}
       69 0003 002D {paintSameArc startangle,endangle}
       FF {fin}
```

```
III.   {CopyBits nopack, norgn, nowoman, nocry}
       GetPenState(pstate);
       SetRect(r, 20, 10, 120, 175);
       SetRect(smallr, 20, 10, 25, 15);
       SetRect(orgr, 0, 0, 30, 20);
       ClipRect(myWindow^.portRect);
       ph := OpenPicture(r);
       PaintRect(r);
       CopyBits (myWindow^.portBits, myWindow^.portBits,
                    smallr, orgr, notSrcXor, NIL);
       {note: result BitMap is 8 bits wide instead of the 5 specified by  smallr}
       ClosePicture;
       SetPenState(pstate); {restore the port's original pen state}
       DrawPicture(ph, r);

data 'PICT' (3)  0048 {size}  000A 0014 00AF 0078 {picFrame}
       1101 {version 1}  01 000A 0000 0000 00FA 0190 {clipRgn — 10 byte region}
       31 000A 0014 00AF 0078 {paintRect rectangle}
       90 0002 000A 0014 000F 001C {BitsRect rowbytes bounds (note that bounds
                    is wider than smallr)}
       000A 0014 000F 0019 {srcRect}
       0000 0000 0014 001E {dstRect}
       00 06 {mode=notSrcXor}
       0000 0000 0000 0000 0000 {5 rows of empty bitmap (we copied from a
                                 still-blank window)}
       FF {fin}
```

Further Reference:

- QuickDraw
- Color QuickDraw
- Using Assembly Language
- Technical Note #59—Pictures and Clip Regions