



#86: MacPaint Document Format

See also: *Inside Macintosh*, Toolbox Utilities

Written by:	Bill Atkinson	1983
Modified by:	Bo3b Johnson	August 19, 1986
Revised by:	Jim Reekes	October 1, 1988

This Technical Note describes the internal format of a MacPaint document, which is a standard used by many programs besides MacPaint. It is the same as the description in the Macintosh Miscellaneous section of early versions of *Inside Macintosh*.

Changes since August 19, 1986: Removed the old clearing routine, added a `NewPtrClear` function, and added a test for corrupted files.

MacPaint documents are easy to read and write and have become a standard interchange format for full-page images on the Macintosh. To help developers generate and read MacPaint documents, we describe their internal format here.

These documents have a file type of “PNTG” and use only the data fork; you may ignore the resource fork. The data fork contains a 512 byte header followed by the compressed data representing a single bit map of 576 pixels wide by 720 pixels tall. At 72 pixels per inch, this bitmap occupies the full 8 inch by 10 inch printable area of the standard ImageWriter printer page.

Header

The first 512 bytes of the document form a header of the following format:

- 4 byte version number (default = 2)
- $38 \times 8 = 304$ bytes of patterns
- 204 unused bytes (reserved for future expansion).

As a Pascal record, the document format could look like the following:

```
MPHeader = RECORD
    Version:      LONGINT;
    PatArray:     ARRAY [1..38] of Pattern;
    Future: PACKED ARRAY [1..204] of SignedByte;
END;
```

If the version number is zero, the document uses default patterns, so you can ignore the rest of the header block, and if your program generates MacPaint documents, you can write out 512 bytes of zero for the document header. Most programs which read MacPaint documents can skip the header when reading.

Bitmap

Following the header are 720 compressed scan lines of data which form the 576 pixel wide by 720 pixel tall bitmap. Without compression, this bitmap would occupy 51840 bytes and chew up disk space pretty fast; typical MacPaint documents compress to about 10 kilobytes using the `_PackBits` procedure to compress runs of equal bytes within each scan line. The bitmap part of a MacPaint document is simply the output of `_PackBits` called 720 times, with 72 bytes of input each time.

To determine the maximum size of a MacPaint file, it is worth noting what *Inside Macintosh* says about `_PackBits`: “The worst case would be when `_PackBits` adds one byte to the row of bytes when packing.” If we include an extra 512 bytes for the file header information to the size of an uncompressed bitmap (51840), then the total number of bytes would be 52352. If we take into account the extra 720 “potential” bytes (one for each row) to the previous total, the maximum size of a MacPaint file becomes 53072 bytes.

Reading Sample

```
{ ReadMPFile:
  This is a small example program written in Pascal that demonstrates
  how to read MacPaint files. As a final step, it takes the data that
  was read and displays it on the screen to show that it worked.
  Caveat: This is not intended to be an example of good programming
  practice, in that the possible errors merely cause the program to exit.
  This is VERY uninformative, and there should be some sort of error handler
  to explain what happened. For simplicity, and thus clarity, those types
  of things were deliberately not included. This example will not work
  on a 128K Macintosh, since memory allocation is done too simplistically.
}

PROGRAM ReadMPFile;

USES MemTypes, QuickDraw, OSIntf, ToolIntf;

CONST
  DefaultVolume = 0;
  MaxUnPackedSize = 51840;           { maximum MacPaint size in bytes }
                                     { 720 lines * 72 bytes/line }

VAR
  srcPtr:      Ptr;
  dstPtr:      Ptr;
  saveDstPtr:  Ptr;
  lastDestPtr: Ptr;
  srcFileName: Str255;
  srcFile:     INTEGER;
  srcSize:     LONGINT;
  errCode:     INTEGER;
  scanLine:    INTEGER;
  aPort:       GrafPort;
  theBitMap:   BitMap;
```

```

FUNCTION NewPtrClear(blockSize: Size): Ptr;
{ This function will return a pointer of size specified and will
  clear the memory to zeros. This is done to create an empty bit map
  containing nothing but white bits.

      MOVE.L      (SP)+,D0          ; get Size variable from stack
      _NewPtr ,clear; make pointer
      MOVE.L      A0,(SP)          ; return pointer
      MOVE.W      D0,MemErr        ; set up MemErr
}
INLINE $201F, $A31E, $2E88, $31C0, $0220;

BEGIN
  { Initialize QuickDraw. }
  InitGraf(@thePort);

  { Make a name of a file to read. }
  srcFileName := 'MP TestFile';

  { Open the data file. }
  errCode := FSOpen(srcFileName,DefaultVolume,srcFile);
  IF errCode <> noErr THEN ExitToShell;

  { Skip the header. }
  srcSize := 512;
  errCode := FSRead(srcFile,srcSize,srcPtr);
  IF errCode <> noErr THEN ExitToShell;

  { Find out how big the file is, and figure out source size. }
  errCode := GetEOF(srcFile,srcSize);
  IF errCode <> noErr THEN ExitToShell;
  srcSize := srcSize - 512; { Remove the header from count. }

  { Make a buffer that is just the right size. }
  srcPtr := NewPtr(srcSize);
  IF srcPtr = NIL THEN ExitToShell;

  { Read the data into the buffer. The file mark is already past the header. }
  errCode := FSRead(srcFile,srcSize,srcPtr);
  IF errCode <> noErr THEN ExitToShell;

  { Close the file we just read. }
  errCode := FSClose(srcFile);
  IF errCode <> noErr THEN ExitToShell;

  { Create a buffer that will be used for the Destination BitMap.
    This also has a maximum size possible. }
  dstPtr := NewPtrClear(MaxUnPackedSize);
  IF dstPtr = NIL THEN ExitToShell;
  saveDstPtr := dstPtr;

  { Unpack each scanline into the buffer. Note that 720 scanlines are
    guaranteed to be in the file. (They may be blank lines) In the
    UnPackBits call, the 72 is the count of bytes done when the file was
    created. MacPaint does one scan line at a time when creating the file.
    The destination pointer is tested each time through the scan loop.
    UnPackBits should increment this pointer by 72, but in the case that
    the packed file is corrupted UnPackBits may end up sending bits into
    uncharted territory. A temporary pointer "lastDstPtr" is used for this
    test.}

```

```

lastDstPtr := dstPtr;
FOR scanLine := 1 TO 720 DO BEGIN
    IF ORD4(lastDstPtr) + 72 <> ORD4(dstPtr) THEN ExitToShell;
    lastDstPtr := dstPtr;
    UnPackBits(srcPtr, dstPtr, 72); {bumps both ptrs}
END;

{ The buffer has been fully unpacked. Create a port that we can draw into. }
OpenPort(@aPort);

{ Create a BitMap out of our saveDstPtr that can be copied to the screen. }
theBitMap.baseAddr := saveDstPtr;
theBitMap.rowBytes := 72; { width of MacPaint picture }
SetPt(theBitMap.bounds.topLeft, 0,0);
SetPt(theBitMap.bounds.botRight, 72*8, 720); {maximum rectangle}

{ Now use that BitMap and draw the piece of it to the screen.
  Only draw the piece that is full screen size (portRect). }
CopyBits(theBitMap, aPort.portBits, aPort.portRect,
        aPort.portRect, srcCopy, NIL);

{ That's it. Now wait for the mouse button to leave. Pause. }
REPEAT
UNTIL Button;
END.

```

Writing Sample

```

{ WriteMPFile:
  This is a small example program written in Pascal that demonstrates how
  to write MacPaint files. It will use the screen as a handy BitMap to be
  written to a file.
}

PROGRAM WriteMPFile;

USES MemTypes, QuickDraw, OSIntf, ToolIntf;

CONST
    DefaultVolume = 0;
    MaxFileSize = 53072;           { maximum MacPaint file size. }

VAR
    srcPtr:      Ptr;
    dstPtr:      Ptr;
    dstFileName: Str255;
    dstFile:     INTEGER;
    dstSize:     LONGINT;
    errCode:     INTEGER;
    scanLine:    INTEGER;
    aPort:       GrafPort;
    dstBuffer:   ByteBuffer;
    I:           LONGINT;
    picturePtr:  Ptr;
    tempPtr:     BigPtr;
    theBitMap:   BitMap;

FUNCTION NewPtrClear(blockSize: Size): Ptr;
{ This function will return a pointer of size specified and will
  clear the memory to zeros. This is done to create an empty file
  buffer containing nothing but white bits.
    MOVE.L      (SP)+,D0          ; get Size variable from stack

```

```

    _NewPtr ,clear; make pointer
MOVE.L      A0,(SP)          ; return pointer
MOVE.W      D0,MemErr        ; set up MemErr
}
INLINE $201F, $A31E, $2E88, $31C0, $0220;

BEGIN
    { Initialize QuickDraw. }
    InitGraf(@thePort);

    { Make an empty buffer that is the picture size. }
    picturePtr := NewPtrClear(MaxFileSize);
    IF picturePtr = NIL THEN ExitToShell;

    { Open a port so we can get to the screen's BitMap easily. }
    OpenPort(@aPort);

    { Create a BitMap out of our dstPtr that can be copied to the screen. }
    theBitMap.baseAddr := picturePtr;
    theBitMap.rowBytes := 72; { width of MacPaint picture }
    SetPt(theBitMap.bounds.topLeft, 0,0);
    SetPt(theBitMap.bounds.botRight,72*8,720); {maximum rectangle}

    { Draw the screen over into our picture buffer. }
    CopyBits(aPort.portBits, theBitMap, aPort.portRect,
             aPort.portRect, srcCopy, NIL);

    { Make a name of a file to write. }
    dstFileName := 'MP TestFile';

    { Create the file, giving it the right Creator and File type.}
    errCode := Create(dstFileName, DefaultVolume, 'MPNT', 'PNTG');
    IF errCode <> noErr THEN ExitToShell;

    { Open the data file to be written. }
    errCode := FSOpen(dstFileName,DefaultVolume,dstFile);
    IF errCode <> noErr THEN ExitToShell;

    { Write the header as all zeros. }
    FOR I := 1 to 512 DO
        dstBuffer[I] := 0;
    dstSize := 512;
    errCode := FSWrite(dstFile,dstSize,@dstBuffer);
    IF errCode <> noErr THEN ExitToShell;

    { Now go into a loop where we pack each line of data into the buffer,
      then write that data to the file. We are using the line count of 72
      in order to make the file readable by MacPaint. Note that the
      Pack/UnPackBits can be used for other purposes. }
    srcPtr := theBitMap.baseAddr; { point at our picture BitMap }
    FOR scanLine := 1 to 720 DO
        BEGIN
            dstPtr := @dstBuffer; { reset the pointer to bottom }
            PackBits(srcPtr,dstPtr,72); { bumps both ptrs}
            dstSize := ORD(dstPtr)-ORD(@dstBuffer);{calc packed size}
            errCode := FSWrite(dstFile,dstSize,@dstBuffer);
            IF errCode <> noErr THEN ExitToShell;
        END;

    { Close the file we just wrote. }
    errCode := FSClose(dstFile);
    IF errCode <> noErr THEN ExitToShell;
END.

```