New Technical Notes

Macintosh

®

Developer Support

MacPaint Document Format

Platforms & Tools   **M.PT.MacPaintDoc**

Revised by: Jim Reekes   June 1989
Written by: Bill Atkinson   1983

This Technical Note describes the internal format of a MacPaint® document, which is a standard used by many other programs. This description is the same as that found in the "Macintosh Miscellaneous" section of early *Inside Macintosh* versions.
Changes since October 1988: Fixed bugs in the example code.

MacPaint documents are easy to read and write, and they have become a standard interchange format for full–page images on the Macintosh. This Note describes the MacPaint internal document format to help developers generate and interpret files in this format.

MacPaint documents have a file type of "PNTG," and since they use only the data fork, you can ignore the resource fork. The data fork contains a 512–byte header followed by compressed data which represents a single bitmap (576 pixels wide by 720 pixels tall). At a resolution of 72 pixels per inch, this bitmap occupies the full 8 inch by 10 inch printable area of a 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*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 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 51,840 bytes and chew

up disk space pretty fast; typical MacPaint documents compress to about 10K 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 (51,840), then the total number of bytes would be 52,352. 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 53,072 bytes.

Reading Sample

```
PROCEDURE ReadMPFile;
{ This is a small example procedure 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.
}

CONST
        DefaultVolume = 0;
        HeaderSize = 512;                  { size of MacPaint header in bytes }
        MaxUnPackedSize = 51840;           { maximum MacPaint size in bytes }
                                           { 720 lines * 72 bytes/line }

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

BEGIN
        errCode := FSOpen('MP TestFile', DefaultVolume, srcFile); { Open the file. }
        IF errCode <> noErr THEN ExitToShell;

        errcode := SetFPos(srcFile, fsFromStart, HeaderSize); { Skip the header. }
        IF errCode <> noErr THEN ExitToShell;

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

```
            srcSize := srcSize - HeaderSize ;        { Remove the header from count. }
            srcPtr := NewPtr(srcSize);               { Make buffer just the right size. }
            IF srcPtr = NIL THEN ExitToShell;

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

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

            { Create a buffer that will be used for the Destination BitMap. }
            dstPtr := NewPtrClear(MaxUnPackedSize); {MPW library routine, see TN 219}
            IF dstPtr = NIL THEN ExitToShell;
            saveDstPtr := dstPtr;

            { Unpack each scan line into the buffer. Note that 720 scan lines 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 where
              the packed file is corrupted UnPackBits may end up sending bits into
              uncharted territory.  A temporary pointer "lastDstPtr" is used for testing
              the result.}

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

            { The buffer has been fully unpacked. Create a port that we can draw into.
              You should save and restore the current port.  }
            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);

            { We need to dispose of the memory we've allocated.  You would not
              dispose of the destPtr if you wish to edit the data.  }
            DisposPtr(srcPtr);                      { dispose of the source buffer }
            DisposPtr(dstPtr);                      { dispose of the destination buffer }
        END;
```

Writing Sample

```
PROCEDURE WriteMPFile;
{ This is a small example procedure 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.
}

CONST
      DefaultVolume = 0;
      HeaderSize = 512;                     { size of MacPaint header in bytes }
      MaxFileSize = 53072;                  { maximum MacPaint file size. }

VAR
      srcPtr:      Ptr;
      dstPtr:      Ptr;
      dstFile:     INTEGER;
      dstSize:     LONGINT;
      errCode:     INTEGER;
      scanLine:    INTEGER;
      aPort:       GrafPort;
      dstBuffer:   PACKED ARRAY[1..HeaderSize] OF BYTE;
      I:           LONGINT;
      picturePtr:  Ptr;
      tempPtr:     BigPtr;
      theBitMap:   BitMap;

BEGIN
      { Make an empty buffer that is the picture size. }
      picturePtr := NewPtrClear(MaxFileSize); {MPW library routine, see TN 219}
      IF picturePtr = NIL THEN ExitToShell;

      { Open a port so we can get to the screen's BitMap easily.  You should save
        and restore the current port. }
      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);

      { Create the file, giving it the right Creator and File type.}
      errCode := Create('MP TestFile', 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;

      FOR I := 1 to HeaderSize DO              { Write the header as all zeros. }
          dstBuffer[I] := 0;
      errCode := FSWrite(dstFile, HeaderSize, @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;

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

Further Reference:
- *Inside Macintosh*, Volume I-135, QuickDraw
- *Inside Macintosh*, Volume I-465, Toolbox Utilities
- *Inside Macintosh*, Volume II-77, The File Manager

MacPaint is a registered trademark of Claris Corporation.