New Technical Notes
Macintosh

®

Developer Support

Displaying Large PICT Files
Imaging

Revised by: March 1988
Written by: Rick Blair July 1987

Now that we have scanners and other massive-picture producing types of applications, there is a need to address the problem of how to display a PICT format object that is bigger than a current PICT resource is allowed to be. Note that this technique applies equally well to version 1 and version 2 (word-opcode) pictures as produced by the Macintosh II.

Future Compatibility

Think of the handle returned by a `GetResource('PICT',ID)` as a "handle" in the more general sense of being an abstract "tag"—something that the ROM routines can use to draw the picture with. Don't assume that the entire picture has been read into memory or that you can directly read any bytes beyond the basic `Picture` record structure (`picSize` followed by `picFrame`). Someday we may provide a mechanism for the resource to be disk- instead of memory-based. The QuickDraw bottleneck procedures will know how to get data from and put data into the pictures in any case.

Spooling from a PICT file

In order to display pictures of arbitrary size, your application should be able to import a QuickDraw picture from a file of type PICT. This is the file 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. See technical note #88 if you are unfamiliar with the `Signal` mechanism. We assume that a `CatchSignals` has been done before `GetandDrawPICTFile` is called.

MPW Pascal Example

```
{the following variable must be at the top level}

VAR
   globalRef   : INTEGER;      {refNum of the file to read from}
```

```
       {the following procedure must be at the top level}
```

```
PROCEDURE GetPICTData(dataPtr: Ptr; byteCount: INTEGER);
{replacement for the QuickDraw bottleneck routine}

   VAR
      err         : OSErr;
      longCount   : LONGINT;

   BEGIN
      longCount := byteCount;
      err := FSRead(globalRef,longCount,dataPtr);
      {can't check for an error because we don't know how to handle it}
   END;

CONST
   abortPICT    = 128;         {error code if DrawPicture aborted}

PROCEDURE GetDrawPICTFile;     {read in a PICT FILE selected by the user}

   VAR
      wher        : Point;     {where to display dialog}
      reply       : SFReply;   {reply record}
      myFileTypes : SFTypeList; {more Standard FILE goodies}
      numFileTypes: INTEGER;

      savedProcs  : QDProcsPtr;
      myProcs     : QDProcs;   {use CQDProcs for a color window}

      myPicture   : PicHandle; {we need a picture handle for DrawPicture}
      longCount   : LONGINT;
      myEOF       : LONGINT;
      myFilePos   : LONGINT;

   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
         SetStdProcs(myProcs); {use SetStdCProcs for a CGrafPort}
         myProcs.getPicProc := @GetPICTData;
         savedProcs := thePort^.grafProcs; {set the grafProcs to ours}
         thePort^.grafProcs := @myProcs;

         myPicture := PicHandle(NewHandle(SizeOf(myPicture)));

         Signal(FSOpen(reply.fname,reply.vRefNum,globalRef));
         Signal(GetEOF(globalRef,myEOF)); {get EOF for later check}
         Signal(SetFPos(globalRef,fsFromStart,512)); {skip header}

         {read in the (obsolete) size word and the picFrame}
         longCount := SizeOf(myPicture);
         Signal(FSRead(globalRef,longCount,Ptr(myPicture^)));

         DrawPicture(myPicture,myPicture^^.picFrame); {draw the picture}

         Signal(GetFPos(globalRef,filePos)); {get position for check}
         Signal(FSClose(globalRef));

         DisposHandle(Handle(myPicture));

         thePort^.grafProcs := savedProcs; {restore the procs}
```

```
                {Check for errors. If there wasn't enough room,}
                {DrawPicture will abort; the FILE position mark}
                {won't be at the end of the FILE.}
                IF filePos <> myEOF THEN Signal(abortPICT);
             END; {IF reply.good}
          END; {GetDrawPICTFile}
```

MPW C Example

```
/*replacement for the QuickDraw bottleneck routine*/
pascal void GetPICTData(dataPtr,byteCount)
Ptr                 dataPtr;
short               byteCount;


{ /* GetPICTData */
       OSErr               err;
       long                longCount;

       longCount = byteCount;
       err = FSRead(globalRef,&longCount,dataPtr);
       /*can't check for an error because we don't know how to handle it*/
} /* GetPICTData */

/*error code if DrawPicture aborted*/
#define         abortPICT     128

OSErr GetDrawPICTFile()           /*read in a PICT FILE selected by the user*/

{      /* GetDrawPICTFile */

       Point         wher;         /*where to display dialog*/
       SFReply       reply;        /*reply record*/
       SFTypeList    myFileTypes;  /*more Standard FILE goodies*/
       short         numFileTypes;
       OSErr         err;
       QDProcsPtr    savedProcs;
       QDProcs       myProcs;      /*use CQDProcs for a color window*/
       PicHandle     myPicture;    /*we need a picture handle for DrawPicture*/
       long          longCount,myEOF,filePos;

             wher.h = 20;
             wher.v = 20;
             numFileTypes = 1;                       /*display PICT files*/
             myFileTypes[0] = 'PICT';
             SFGetFile(wher,'',nil,numFileTypes,myFileTypes,nil,&reply);

             if (reply.good)
              {
                    SetStdProcs(&myProcs);
                    /*use SetStdCProcs for a CGrafPort*/
                    myProcs.getPicProc = GetPICTData;
                    savedProcs = (*qd.thePort).grafProcs;

                    /*set the grafProcs to ours*/
                    (*qd.thePort).grafProcs = &myProcs;

                    myPicture = (PicHandle)NewHandle(sizeof(Picture));

                    err = FSOpen(&reply.fName,reply.vRefNum,&globalRef);
                    if (err != noErr) return err;

                    err = GetEOF(globalRef,&myEOF);
                    /*get EOF for later check*/
                    if (err != noErr) return err;

                    err = SetFPos(globalRef,fsFromStart,512);/*skip header*/
                    if (err != noErr) return err;

                    /*read in the (obsolete) size word and the picFrame*/
                    longCount = sizeof(Picture);
                     err = FSRead(globalRef,&longCount,(Ptr)*myPicture);
                    if (err != noErr) return err;

                    DrawPicture(myPicture,&(**myPicture).picFrame); /*draw
                                        the picture*/

                    err = GetFPos(globalRef,&filePos);/*get position for
                                 check*/
                    if (err != noErr) return err;
                    err = FSClose(globalRef);
                    if (err != noErr) return err;

                    DisposHandle((Handle)myPicture);

                    (*qd.thePort).grafProcs = savedProcs;/*restore the
                                 procs*/

                     /*Check for errors. if there wasn't enough room,*/
                     /*DrawPicture will abort; the FILE position mark*/
                    /*won't be at the end of the FILE.*/

                    if (filePos != myEOF)  return abortPICT;
                    else return noErr;
             } /*if (reply.good) */
}      /* GetDrawPICTFile */
```

More on Picture 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 was 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.

If you do look at the picture data be sure and check the version information. You may want to put up an alert in your application that indicates to the user when a picture was created using a later version of the picture format than your application recognizes, 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 ones it knows.

As with reading picture data directly, it is best to use QuickDraw to create data in the PICT format. If you do need to create PICT format data directly, it is essential that you use the latest opcode specifications and that you thoroughly test the data produced on both color and black and white Macintosh machines. Contact Macintosh Developer Technical Support if you are not sure that you have the latest specifications.

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

Further Reference:

- QuickDraw
- Technical Note M.IM.PictureOpcodes —
  Internal Picture Format
- Technical Note M.PT.Signals —
  Signals