Technical Note PR515
Printing Manager Q&As

CONTENTS

This Technical Note contains a collection of archived Q&As relating to a specific topic--questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&A's can be found on the Macintosh Technical Q&A's web site.

[Oct 01 1990]

OK to call WaitNextEvent if modal window is frontmost

Date Written: 1/20/93

Last reviewed: 6/14/93

I've read in "Print Hints: Top 10 Printing Crimes" that a program shouldn't call `WaitNextEvent` while the Print Manager is open, but I call `WaitNextEvent` in my pIdleProc so my application can detect a user cancellation. Could this cause a problem?

The reason you shouldn't call `WaitNextEvent` is to prevent the system from switching to another application. If you have a modal dialog-type window in front (the one with the modal window proc ID), the system will never switch applications and it's safe to call `WaitNextEvent`. This is why all Apple's drivers that call `WaitNextEvent` put up a modal dialog box. If you have a modal status window frontmost when your `pIdle` procedure is called, you're in no danger.

Back to top

Macintosh Printing Manager error -8132

Date Written: 1/1/90

Last reviewed: 6/14/93

What is Macintosh Printing Manager error -8132?

Error -8132 is generated when the selected printer times out. AppleTalk printers monitor the time between data packets that are sent to them. These devices will wait a maximum of two minutes without data until generating a time-out error. Applications such as database report generators sometimes require longer than two minutes to sort or format their data. These applications may receive the -8132 error. One workaround for this problem is to spool the report to a file that can be printed when sorting/formatting is completed.

Another possible solution is to change the amount of time the LaserWriter waits for data until it times out. Keep in mind that the time-out in the LaserWriter exists for a reason. Say a user starts printing a large document and leaves for the day. About halfway through the document, the user's Macintosh crashes, leaving the printer waiting without data. In two minutes, the LaserWriter times out, and is once again available for other users on the net. However if the time-out is disabled by the first user's job, the printer is useless until rebooted. To avoid disabling the time-out, increase the time-out value so that the device times-out if the job takes too long to print, but does not time-out for normal delays. This is done by sending the following Postscript to the LaserWriter with a Postscript downloading utility:

```
serverdict begin 0 exitserver
```

See the Apple's LaserWriter Reference for more information.

The only other way to handle this problem is to periodically send some data to the printer. Do this about once a minute or so to prevent the printer from timing out. Use the PostScriptHandle `PicComment` to send this "tickle" data so that it is only sent to PostScript printers that have the ability to time-out. (Quickdraw printers won't see the data.) Another reason for using the PostScriptHandle `PicComment` involves the internal buffering of the LaserWriter driver. The LaserWriter driver maintains a 4K internal buffer for the PostScript that it sends. If you send just a few characters as your tickle data, they are buffered in the driver, and the device still times-out. The only way to ensure that your tickle data makes it to the device is to send a 4K block via the PostScriptHandle `PicComment`. Needless to say, sending 4K blocks over AppleTalk once a minute is going to cause some network traffic, but it's still another possible workaround.

Back to top

Macintosh Printing Manager error -8133

Date Written: 1/1/90

Last reviewed: 8/1/92

What is Macintosh Printing Manager error -8133?

Printing Manager error -8133 occurs when the PostScript interpreter of the LaserWriter (or any other PostScript printer) generates a PostScript error. A description of the PostScript command that caused the error is displayed in the status window. This error often occurs when an application sends PostScript directly to the printer, and that PostScript contains an error. To debug this problem, look at the PostScript generated by the driver. Hold down the Command-F key right after clicking OK in the Print dialog. A file named PostScript0 will be created in the current directory. With the 7.x LaserWriter drivers, just select the PostScript File radio button in the Print dialog.

Back to top

Identifying Macintosh's currently selected printer

Date Written: 1/1/90

Last reviewed: 8/1/92

How can I find out which printer the user has selected with the Macintosh Chooser? How can I change it without going through the Chooser?

The type of the currently selected printer is stored in the System file in `'STR '` resource ID -8192. This will be the name of the driver selected, such as LaserWriter or ImageWriter. The name of the actual printer selected will be in the driver file (same name as the printer type chosen) in `'PAPA'` resource ID -8192. Look at this resource with ResEdit to see what is in it.

You can change the `'STR '` and `'PAPA'` resources, but it may not work now or in the future. *DTS does not recommend changing the system file or circumventing the Chooser!* The Chooser is a very complicated piece of software, and it has many dependencies. Implementing the chooser within your application would make your application dependent on a particular version of the system software. Like the Chooser, your application would then have to be revised each time a new system is released.

Back to top

How to save Macintosh Page Setup and Print dialog options

Date Written: 5/3/89

Last reviewed: 6/14/93

How do I save options set in the Macintosh Page Setup and Print dialogs?

The only supported method to save the dialog options is to save the Print Record that was passed to the `PrStlDialog/PrJobDialog` calls. This print record contains the options chosen in the Page Setup and Print dialogs. Although these options are saved in the print record, they cannot be used as defaults for the Print dialog. For example, say the user chooses landscape printing in the Page Setup dialog, and sets the number of copies to 5 in the Print dialog. If the print record is passed to `PrStlDialog`, the dialog shows that landscape printing was selected in the print record. However, if that print record is passed to PrJobDialog, number of copies is shown as 1, because options in the PrJobDialog are considered to be "job dependent," as opposed to "document dependent."

The best method for saving the print record is to save it as a resource in your document's resource fork. Since the print record is already pointed to by a valid handle, creating a resource is easy. Here are the required steps:

1. Save the refNum returned by CurResFile;
2. Make sure the resource fork of the document file is opened using OpenResFile (IM I:115).
3. Now that you have a place to put the resource, you must get the resource manager to create it for you. First, pick a type and ID for your resource. You can use the Unique ID function (IM I:121) to generate the ID. Next, call the AddResource procedure (IM I:124). This procedure takes four parameters:

```
Parameter  Type     What to Pass
-------------------------
theData    Handle   The handle to the currently valid print record.
theType    ResType  The four character resource type you have chosen.
theID      INTEGER  The ID returned from the UniqueID function.
```

4. Call WriteResource (IM I:125), and UpdateResFile (IM I:125) to make sure the new resource gets saved into the file.
5. Reset the currently selected file back to whatever it was before you began the save using UseResFile (IM I:117).

The important things to keep in mind when doing this are:

1. Make your resource type something different than those used by the Print Manager. This prevents the Print Manager from getting confused and grabbing the wrong resource. Types to avoid include 'PREC', 'PDEF', and 'POST'.
2. Don't make any assumptions about the size of the print record. If you really need to know the size, use GetHandleSize so that if the record gets bigger in the future, your code will still work.
3. Be sure to pass the record that you get from your document to PrValidate before using it. This ensures that the record gets updated if any changes have been made to record structure or contents since the record was saved.

Back to top

MacApp and SetRsl print driver resolution

Date Written: 4/16/92

Last reviewed: 6/14/93

We've been trying to use the standardPrint handler from MacApp 2 to print some views. The problem is that we would like the printout to come out at 300 dpi on the LaserWriter, and the print handler's device resolution fields (which come from the device driver) tell it to draw at 72 dpi. How, can I get a PenSize(1,1); MoveTo(100,100); LineTo(300,300); draw at 300 dpi and make a nice fine line? Please refer me to the needed documentation.

The mechanism by which your application can find out the chosen printer's real resolution, and then draw with it, is documented "Meet PrGeneral, the Trap That Makes the Most of the Printing Manager," in issue #3 of *develop* magazine (July 1990). You can find it, along with its accompanying code and application, on the Developer CD, E.T.O., and AppleLink.

The DTS Sample Code SC.009.FracApp300 on the Developer CD shows how to use PrGeneral in an application built with MacApp 2.

Back to top

Printing patterns at higher resolutions with GetRslData & SetRsl

Date Written: 11/7/90

Last reviewed: 6/14/93

Is there a way that I can make my Macintosh fill patterns print at a higher resolution (more than 72 dpi), but have my patterns still print as patterns?

To make your patterns print at the printer's resolution, you need to use Printing Manager PrGeneral's GetRslData and SetRsl opcodes to get and set the resolution, and you must scale the pattern to match. Let me explain.

If we do not scale our patterns up to the printer's resolution before print time, we would get "big chunky" patterns because the printer driver would need to scale the patterns on the fly from 72 dpi to its resolution. Therefore, we use the "cookie cutter" approach to "place" the pattern into the object that is being filled. The size of the cookie cutter (that is, the destination Rect) depends on the "scaleFactor". For example, a scaleFactor of 2 will have a destination rect of 16

x 16. We will then CopyBits the pattern one square at a time into the object that is being filled.

XRefs:

Inside Macintosh Volume V, pages 410-416

"Meet PrGeneral," *develop,* July 1990

Back to top

Call Printing Mgr's bottleneck to send StdText to printer driver

Date Written: 11/7/90

Last reviewed: 6/14/93

We need to use the Macintosh toolbox call StdText instead of DrawChar, DrawString, and DrawText to accomplish horizontal scaling and fractional point sizes, but StdText does not work on the LaserWriter IISC. Is there any workaround?

The problem is that you are calling the low-level bottleneck directly, and the Printing Manager has replaced the bottleneck completely with its own. So when you are actually printing, the print driver will not see your call to StdText. The work-around for this is to call the Print Manager's bottleneck itself when you are printing. You can do this in the following way:

```
/* typedef the function */
typedef pascal void (*StdTextProcPtr)( short, Ptr, Point, Point );

void tstDrawStuff( pInfo, thePort, pageNum )
TPrint pInfo;
GrafPtr thePort;
unsigned short pageNum;
{
   StdTextProcPtr Proc;     /* declare a variable here */

/* do your clipping etc. */

/* TEST -- no good w/o PriMon */
/* do your moveto etc. */

/* Is there a replacement bottleneck ?*/
if ( thePort->grafProcs )  {
    Proc = (StdTextProcPtr)thePort->grafProcs->textProc; /* yes, so grab it */
    (*Proc)( strlen("HELLO") , (Ptr) "HELLO", numer, denom );
}
/* else just call standard text */
```

This technique will work in both background and foreground printing. I have tested this solution on the Personal LaserWriter SC and the LaserWriter IINTX, and it prints fine.

Back to top

Modifying PDEFs to have printer driver jump into dialog code

Date Written: 11/9/90

Last reviewed: 6/14/93

Could you please point me to the correct section in "Learning to Drive" (or send me an example) that we would need to modify the appropriate PDEF resource in order to have the driver jump to our dialog code?

Pages 15-18 discuss the contents of the various PDEFs. You will need patch the appropriate PDEF to get your required functionality. You should also realize that patching PDEFs is not an easy task, nor will it always survive "new" Macintosh LaserWriter driver releases. I want you to know that before you go too far down the PDEF patching road.

By the way, we do know of a couple of developers that have patched PDEFs successfully. The following paragraphs are a "high-level" look at patching a PDEF:

To patch a PDEF, you must write an installer application that lets users choose the printer driver they want to modify via Standard File. You can cause Standard File to display only printer drivers by telling it to display only files of type "PRER". Once the user has selected a driver you need to scan the driver for the PDEF 0 and 1 resources. When you find one of these you read it in and modify it.

However, I'll give you a quick overview of what you need to do. PDEF 0 & 1 both start with jump tables (a table of JMP instructions followed by the PC relative address to jump to). The JMP to PrClosePage always begins at offset 12. So what your installer needs to do after it loads the resource is grab the eighth word in the resource. That is the offset to the actual start of the PrClosePage code. Then you have to resize the handle so that it is big enough to hold your patch and then BlockMove {IM II-44} your code into the end of the handle. The following illustration might help:

```
         JMP $0486              << Note: some PDEF's have multiple JMPs at
* - - - - - - - - - - - - - *            the beginning of the code. The
*                           *            original offset to PrClosePage was
* >> Original PDEF code  << *            $0386.
*                           *
$0386 --> PrClosePage           << Note: most PDEF's have multiple printing
*                           *            calls defined.
*                           *
* - - - - - - - - - - - - - *
$0486 --> {{{  Your Code  }}}
*                           *   << You resize the resource handle to this
*                           *            and BlockMove your code in here.
*                           *
* - - - - - - - - - - - - - *
         JMP $0386              << At this point we jump into the printer
```

Note:
The previous illustration is not to scale nor completely accurate in regards to the addresses used. Please consult "Learning To drive" for the correct offsets for each PDEF.

Then replace the original offset to PrClosePage with the offset to your new code. Remember that this is PC-relative so you would calculate it like this: If the original PrClosePage JMP was JMP 10(SP) and the size of the original PDEF was 50 bytes, then you need to replace the JMP 10(SP) with JMP 50-16(SP). In other words, you need to JMP to what was formerly the end of the PDEF resource but what is now the beginning of your code (subtract 16 because the offset table is 16 bytes long. Refer to "Learning to Drive" for more details). Then on the end of your code, add a JMP ((SizeOfJumpTable + originalOffset) - (sizeofOriginalPDEF + SizeOfYourCode))(SP). Your code will end with a JMP to a negative offset off the PC back to the original PrCloseRoutine.

If you are willing to support only system software later than version 3.3 (basically that means that you won't be supporting Macintosh 512K and 512KE computers), you don't have to deal with all the PDEF changing, and you can simply patch out the $A8FD trap and look for the proper selector on the stack. (Use SysEnvirons to be sure that you are on a system later than version 3.3) This will work because the newer libraries (that is, any library that shipped since the Macintosh II shipped) first checks for the trap and jumps there if the trap exists so even if the application has linked in the printing library they still end up calling the trap on systems after 3.3.

Back to top

Determining if printer supports color and/or grayscale printing

Date Written: 3/14/91

Last reviewed: 8/1/92

Is there any way to determine whether I'm printing to either a color printer or a printer simulating color, such as the LaserWriter set for Color/Grayscale?

Check the grafPort returned by your call to PrOpenDoc. If the rowBytes of the grafPort is less than 0, the Printing Manager has returned a color grafPort. You can then make Color QuickDraw calls into this grafPort. LaserWriter driver version 6.0 and later returns a color grafPort from the PrOpenDoc call, if the Color/Grayscale button has been set.

The following code fragment demonstrates this:

```
(* This function determines whether the port passed to it is a *)
(* color port. If so, it returns TRUE.                          *)
FUNCTION IsColorPort(portInQuestion: GrafPtr): BOOLEAN;
BEGIN
  IF portInQuestion^.portBits.rowBytes < 0 THEN
    IsColorPort := TRUE
  ELSE
    IsColorPort := FALSE;
```

A third-party printer driver should return the type of `grafPort` that represents the abilities of its printer. Therefore, if the printer supports color and/or grayscale, and if Color QuickDraw is present, the application will receive a color `grafPort` after calling `PrOpenDoc`. Otherwise, if the Macintosh you're running on does not support Color QuickDraw, you should return a black-and-white `grafPort`.

Back to top

Using LaserWriter fonts with StyleWriter

Date Written: 3/22/91

Last reviewed: 6/14/93

Can StyleWriter use all the Adobe fonts for my LaserWriter? Can StyleWriter print the encapsulated PostScript drawings I've developed? Does TrueType software come with the StyleWriter?

The difference between the StyleWriter and a regular LaserWriter (other than one's got a laser and the other doesn't) is that the LaserWriter has PostScript built-in; the StyleWriter does not. The StyleWriter, in fact, has nothing built-in, and only recognizes basic "put this dot here" types of commands. Even sending it pure ASCII accomplishes nothing.

TrueType software is shipped with the StyleWriter, because it can image fonts at any resolution with excellent quality, much like PostScript. The TrueType software is intended for users of system versions 6.0.7 and later, but prior to System 7. System 7 *includes* support for TrueType.

You can use PostScript images and fonts with the StyleWriter, but just as with the ImageWriter, you'll have to have something that images them in memory before sending them to the printer (in other words, a Macintosh-resident PostScript interpreter).

Back to top

Personal LaserWriter LS 1.1 driver is faster

Date Written: 8/9/91

Last reviewed: 6/14/93

My LaserWriter LS is very slow when printing from my Macintosh SE. Any suggestions?

Use the new LaserWriter LS driver version 1.1 or later. It was released after System 7.0, and is available on AppleLink or from your dealer.

If you're printing Word files, Microsoft's TrueType INIT will help. It's available from MicroSoft Technical Support or from commercial bulletin board services.

Back to top

ImageWriter II 6.1 & 7.0 driver PrGeneral/PrStlDialog bug

Date Written: 9/11/91

Last reviewed: 6/14/93

If an application uses `PrGeneral` to set `draftBits` mode on an ImageWriter II (using the 6.1 or 7.0 drivers) and then does a `PrStlDialog`, a fatal crash occurs. This happens with both the AppleTalk and non-AppleTalk drivers. This only happens on Macintosh systems with Color QuickDraw.

The problem is connected to the two `'dctb'` resources which were added in the 6.1 driver. When the landscape icon is being grayed out (since it's unsupported in `draftBits` mode), the driver is jamming the gray pen pattern directly into

the grafPort. While this brute-force method works fine with GrafPorts, the 'dctb' resources make the Dialog Manager use a CGrafPort now. Jamming the pattern into this CGrafPort as if it were a grafPort makes a mess and causes a fatal crash, manifesting itself as a "division by zero" error, for example.

To work around the problem, check to see if you are printing to a 6.1 or 7.0 ImageWriter driver, and if so, do the style dialog before setting draftBits. The disadvantage of this is that setting draftBits before calling PrStlDialog causes the 50% reduction and landscape controls to be disabled, and this method doesn't. Therefore, after the style dialog, you should check to see if the user selected landscape printing or 50% reduction by calling PrGeneral (getRotnOp) and checking bit 3 of the wDev field (See "Learning To Drive" on the Developer CDs for more about the wDev.) If the user selected either of those options, display a dialog warning them that those features will be ignored. Then, call PrGeneral to set draftBits and continue normally. PrGeneral will adjust the print record so that options not supported with draftBits are turned off. If a non 6.1/7.0 ImageWriter driver is used, the code should set draftBits as usual, before the PrStlDialog call.

This workaround is the better alternative to hacking the 'dctb's out of the driver or patching around the problem, although it doesn't help 7.0 users running applications that already make use of the draftBits option. The following Pascal snippet demonstrates how to use the workaround:

```
(* Sample snippet that demonstrates how to work-around the
   'dctb' bug in the ImageWriter II 6.1 and 7.0 drivers.
   Bug causes a crash when doing a style dialog
   after setting draftBits with PrGeneral.
 *)

{
  What's what in the code snippet that follows.

  thePrRecHdl        : THPrint;
  rslData            : TSetRslBlk;
  draftData          : TDftBitsBlk;
  rotnData           : TGetRotnBlk;
  isBadImageWriter   : Boolean;
  isLandscape        : Boolean;
  isReduced          : Boolean;
}

      PrOpen;

    IF (PrError = noErr) THEN
    BEGIN

      PrintDefault(thePrRecHdl);

      IF (PrError = noErr) THEN
      BEGIN

{See if we're using a version 6.1 or 7.0 ImageWriter driver.}

        IF (thePrRecHdl^^.prStl.wDev DIV 256) = 1 THEN
          isBadImageWriter := (PrDrvrVers = 61) OR (PrDrvrVers = 70)
        ELSE
          isBadImageWriter := FALSE;

        IF NOT (isBadImageWriter) THEN
        BEGIN

{Set draftBits mode now, if not our special case.}

          draftData.iOpCode := draftBitsOp;
          draftData.hPrint := thePrRecHdl;
          PrGeneral(@draftData);

        END;

{Do the style dialog.}

        IF (PrStlDialog(thePrRecHdl)) THEN
          IF (isBadImageWriter) THEN
          BEGIN
```

```
{If so, see if they selected rotation or 50% reduction.}

            rotnData.iOpCode := getRotnOp;
            rotnData.hPrint := thePrRecHdl;
            PrGeneral(@rotnData);

            isLandscape := rotnData.fLandscape;

            isReduced := BTst(thePrRecHdl^^.prStl.wDev, 3);

{If so, warn them that those things will be ignored.
 Preferably use something other than DebugStr...      }

            IF (isLandscape) OR (isReduced) THEN
              DebugStr('Bad ImageWriter; landscape and/or reduced.');

{Now, set draftBits mode, since we didn't before.}

            draftData.iOpCode := draftBitsOp;
            draftData.hPrint := thePrRecHdl;
            PrGeneral(@draftData);

          END;

          IF (PrJobDialog(thePrRecHdl)) THEN
          BEGIN

            {Print}

          END;
      END;
    END;
```

Back to top

How to abort printing while PrPicFile is executing

Date Written: 9/27/91

Last reviewed: 6/14/93

Can I abort printing from within `PrPicFile`? My attempts haven't been succesful.

The following Pascal procedure is *roughly* the same as the default idle procedure--the one the system supplies for you when you don't supply one of your own:

```
procedure MyPIdleProcedure;
  var
    pIdleEventRecord: EventRecord;
  begin
    if GetNextEvent(keyDownMask, pIdleEventRecord) then
      begin
        if ((BitAnd(pIdleEventRecord.message, charCodeMask) =
          longint('.')) and
        (BitAnd(longint(pIdleEventRecord.modifiers), cmdKey) =
          cmdKey)) then
            PrSetError(iPrAbort);
      end;
```

(This is written in Think Pascal, but it should work fine under MPW Pascal if you change the "`BitAnd`" functions to "BAND".)

As you can see, the system aborts when you press Command-period by using `PrSetError` to the constant iPrAbort, which is the same error returned when the user clicks "Cancel" in the style or job dialogs. If you set the error to `iPrAbort`, the system will take care of the rest. Your print loop will also gracefully cancel, provided you follow the guidelines in the Macintosh Technical Note "A Printing Loop That Cares."

Back to top

Disabling ImageWriter's initial formfeed

Date Written: 1/23/92

Last reviewed: 6/14/93

When our product-to-be prints to the ImageWriter II there is an automatic (and undesirable in our product) sheet feed before actual printing begins. To make matters worse, it always feeds 11", even though our form is a different length. Is there a way to avoid this?

The ImageWriter ejects a page before printing with the "No Gaps" option selected because it assumes that the page is aligned with a small gap between the perforation and the print head. So the printer ejects the page and re-aligns the print head at the perforation of the next page.

The sample code below illustrates how to prevent that nasty form feed from occurring. The program temporarily disables the "No Gaps" option if it is selected until after PrOpenDoc is called, preventing a formfeed from occurring. After PrOpenDoc is called, the program re-enables the "No Gaps" option.

Please note that this sample only works with the ImageWriter II and the ImageWriter LQ printers--probably the only series that supports the "No Gaps" option. The sample shows how to check for these types of printers, so you shouldn't have any problems.

```
PROGRAM NoGapsWA;

{$U-}
USES PasInOut,Memtypes,QuickDraw,OSIntf,ToolIntf,PackIntf,MacPrint;

CONST
  bDevLaser = 3;
  bDevImageWriter     =   1;
  bDevImageWriterLQ   =   5;

PROCEDURE DrawStuff(theWorld : Rect);
  { Draw whatever you want here. Make sure it fits in the world rectangle. }
BEGIN
  MoveTo(100, 100);
  DrawString('testing...');
END;

FUNCTION HiByte(word: INTEGER): Byte;
BEGIN
  HiByte := word DIV 256;
END;

PROCEDURE PrintStuff;
VAR
  thePrRec:   THPrint;
  thePrPort:  TPPrPort;
  theStatus:  TPrStatus;
  oldPort:    GrafPtr;
  theError:   OSErr;
  theVers:    INTEGER;
  NoGaps:     BOOLEAN;

BEGIN
  GetPort(oldPort);
  thePrRec := THPrint(NewHandle(SIZEOF(TPrint)));
  PrOpen;
  IF PrError = noErr THEN BEGIN
    PrintDefault(thePrRec);

    IF NOT PrStlDialog(thePrRec) THEN
      PrSetError(iPrAbort);
    IF NOT PrJobDialog(thePrRec) THEN
      PrSetError(iPrAbort);

    (* First, lock the handle down so we can do a cool with statement. *)
```

```
    HLock(Handle(thePrRec));
    IF PrError = noErr THEN
    WITH thePrRec^^ DO BEGIN
      (* Okay, now get device dependent.  We only want to do this if we *)
      (* know the selected printer driver supports it.                  *)
      IF (HiByte(prStl.wDev) = bDevImageWriter) OR
      (HiByte(prStl.wDev) = bDevImageWriterLQ) THEN BEGIN

        (* Now remember the state of the nogaps option.          *)
        NoGaps := BitTst(@prStl.wDev, 11);

        (* If NoGaps was selected, then turn it off until we PrOpenDoc.*)
        IF (NoGaps) THEN BitClr(@prStl.wDev, 11);

        (* If we're on the LQ driver, we need to copy the wDev into the*)
        (* PrintX array, or the LQ driver won't notice that it changed.*)
        IF (HiByte(prStl.wDev) = bDevImageWriterLQ) THEN
          PrintX[2] := prStl.wDev;
      END;
    END;
    HUnlock(Handle(thePrRec));

    (* Now open the document. *)
    thePrPort := PrOpenDoc(thePrRec, NIL, NIL);

    (* Now we need to see if the NoGaps option was selected. If it was *)
    (* then we turned it off before calling PrOpenDoc, so we need to   *)
    (* turn it back on. If wasn't selected, we won't do anything.      *)
    HLock(Handle(thePrRec));
    WITH thePrRec^^ DO BEGIN
      IF NoGaps AND
        (HiByte(prStl.wDev) = bDevImageWriter) OR
        (HiByte(prStl.wDev) = bDevImageWriterLQ) THEN BEGIN
        (* Turn it on for the ImageWriter. *)
        BitSet(@prStl.wDev, 11);

        (* Copy it over for the ImageWriter LQ. *)
        IF (HiByte(prStl.wDev) = bDevImageWriterLQ) THEN
          PrintX[2] := prStl.wDev;
      END;
    END;
    HUnlock(Handle(thePrRec));
    IF PrError = noErr THEN BEGIN
      PrOpenPage(thePrPort, NIL);
      IF PrError = noErr THEN BEGIN
        DrawStuff(thePrRec^^.prInfo.rPage);
      END;
      PrClosePage(thePrPort);
    END;
    PrCloseDoc(thePrPort);
    IF (thePrRec^^.prJob.bJDocLoop = bSpoolLoop) and (PrError = noErr)
      THEN PrPicFile(thePrRec, NIL, NIL, NIL, theStatus);
  END;
  PrClose;

  DisposHandle(Handle(thePrRec));
  SetPort(oldPort);
END;

BEGIN
  InitGraf(@thePort);                          {initialize QuickDraw}
  InitFonts;                                   {initialize Font Manager}
  FlushEvents(everyEvent, 0); {call OS Event Mgr to discard}
                              {any previous events}
  InitWindows;                                 {initialize Window Manager}
  InitMenus;                                   {initialize Menu Manager}
  TEInit;                                      {initialize TextEdit}
  InitDialogs(NIL);                          {initialize Dialog Manager}
  InitCursor;           {call QuickDraw to make cursor (pointer) an arrow}

  PrintStuff;
```

Back to top

Update Macintosh color table if printing in color

Date Written: 5/3/89

Last reviewed: 6/14/93

I'm having trouble printing in color. When I copy a color image from a Macintosh offscreen pixmap to the printer, I get a black piece of paper. If I draw directly it comes out fine. What am I doing wrong?

When you built your offscreen port and copied the color table from a `GDevice`, did you change the color table to reflect that the color table is now part of a pixmap? In a GDevice the color table's value field is zero for all entries, but in a pixmap the value field represents the index value of each color in the table. If the color table has not been converted, printing won't work properly.

Back to top

Downloadables

Acrobat version of this Note (K). Download