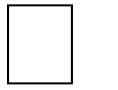
New Technical Notes

Macintosh



Developer Support

Programmatic PostScript Files

Imaging

Written by: Matt Deatherage & Hugo Ayala September 1992

This Technical Note discusses how to make the LaserWriter driver, versions 7.0 and later, create PostScript files from your printing loop, and when this is and is not appropriate.

Topics

- Creating PostScript files
- The LaserWriter driver is not a PostScript conversion utility
- Sample functions for creating an appropriate print record

I get to make PostScript files?

Yes, but don't get too excited yet. The feature we're about to describe is not applicable in every case where you might want to generate PostScript code, so it's not a generic solution. Like those standardized tests used to say, "Please read all the directions before marking the page."

There are definitely situations where forcing the LaserWriter driver to create a PostScript file has advantages. It's extremely handy for many large, multi-network Macintosh installations that have unique printing needs. It can be handy for people using non-AppleTalk PostScript printers. It's a very useful capability in this and dozens of other situations.

What it's **not** useful for is generic PostScript conversion.

The LaserWriter driver was designed expressly for the purpose of converting QuickDraw calls to PostScript for high-resolution imaging on Apple's PostScript printers. As the years have gone by, it's been adapted to things that weren't part of the original design model (like driving non-Apple PostScript printers). One of the features added in recent years was the capability to dump a PostScript file when printing. This was originally only a debugging feature for the convenience of the engineers working on the driver, but when it was made accessible to users through a hidden dialog item and a not-so-secret key combination, the response was overwhelming. In response, the printing engineers included the capability to pick a PostScript file for output was built into versions 7.0 and later of the driver.

That didn't change the fact that it was still, at heart, a glorified debugging feature. The

PostScript the driver creates is designed expressly for printing. It does not create Encapsulated PostScript files (EPSF format). It doesn't easily incorporate any PostScript document conventions other than the one it uses for its own benefit. It doesn't generate PostScript Level 2. It includes large amounts of custom encrypted code to provide TrueType™ font rendering technology on PostScript printers where possible.

It's important to understand that you can't change any of this. If you have a strong need to programmatically create files just like the LaserWriter driver would create if you chose the "PostScript file" radio button in the job dialog, this technique is for you. If you need anything more customizable, or more specialized, or in a different format, this Note is not for you. The LaserWriter driver is not and never has been a generic PostScript conversion utility; if you want such capabilities, you must still build them into your application.

How to Do the Deed

The LaserWriter driver stores the PostScript file information in the print record—but not in a format designed for your easy access. To make it print to a PostScript file, you have to set a bit, fill in a `vRefNum` , a `dirID` and a *pointer* to the file name—all inside a private LaserWriter driver structure.

The flag is a bit in the word starting at offset $52 (or 82 decimal) of the print record. This is documented as boolean value `f2` of the printer flags record `prFlag1`. Figure 1 shows a redeclaration of this record with the newly-documented `fPSFile` bit included.

```
lwTPrFlag1 = PACKED RECORD
                f15: BOOLEAN;
                f14: BOOLEAN;
                f13: BOOLEAN;
                f12: BOOLEAN;
                f11: BOOLEAN;
                f10: BOOLEAN;
                f9: BOOLEAN;
                f8: BOOLEAN;
                f7: BOOLEAN;
                f6: BOOLEAN;
                f5: BOOLEAN;
                f4: BOOLEAN;
                f3: BOOLEAN;
                fPSFile: BOOLEAN;
                fLstPgFst: BOOLEAN;
                fUserScale: BOOLEAN;
             END;
```

Figure 1—Redefined PrFlag1 Record

Note that the record is renamed beginning with `lw` to emphasize that this technique **only** works with the LaserWriter driver. This means you have to check that you're printing to the LaserWriter driver by making sure the high byte of `TPRStl.wDev` is 3, for the LaserWriter driver. If other drivers impersonate the LaserWriter driver (which they may do, for example, so that your application sends them PostScript in picture comments), this **may** crash. Unfortunately, Apple can't conclusively tell you that a third-party printer driver will or will not support this feature. Apple's driver does.

Where does the LaserWriter driver put the new PostScript file? The word starting at offset $68 (104 decimal) is the `vRefNum` for the file to save, and the long at offset $70 is the directory ID for where to store the file. The long starting at $6A (or 106 decimal in the print record) is a

pointer to the PostScript file name as a Pascal string—just the file name; the other fields indicate the other portions of the file's specification (in fact, they're all three portions of an `FSSpec` record).

However, the name pointer isn't just any old pointer. There's some baggage with it.

First, the LaserWriter driver always assumes that it allocated the pointer, and somewhere in the printing loop, when it's done with the PostScript file name, it calls `DisposPtr` on that pointer. That means you **must** allocate it with `NewPtr` and it **must** be valid during the print loop, or your heap becomes toast. It should be a 64-byte pointer, because the driver assumes that's what it is.

Second, the LaserWriter driver assumes that it's the only one who would be messing with this private field in the print record, so it only clears it during `PrJobInit`. That means when you're done printing, your print record will still contain a pointer to the memory that used to point to the file name string. If you save this print record and try to print again without going through the job dialog, the LaserWriter driver will use it as a pointer on your behalf, even though it probably doesn't point to anything useful anymore. Also, if the pointer is non-`NIL` entering the job dialog (if somehow someone bypasses `PrJobInit`), it also assumes it's a pointer and uses it. So, before you save or use this print record for anything, **always** zero the file name pointer. It will save many hassles down the road.

Third, if you encounter a print record that has the `fPSFile` bit (bit 3 of offset $52) already set, the LaserWriter driver has already validated the information in the print record. The pointer has been allocated and the directory information stored. Do **not** override these values or you may cause a memory leak.

The driver also saves in the print record, in the next two bytes starting at $6E, the `vRefNum` of the last volume used to store a PostScript file. It also uses the directory ID field at offset $70 to indicate the last directory used for storing a PostScript file. This is only so the Standard File dialog put up by `PrJobDialog` can start with the most appropriate directory; it's not important to programmatic control of this mechanism.

Figure 2 shows the LaserWriter driver's version of the print record, redeclared with the newly-documented information as `lwTHPrint`:

```
TYPE
  { Print Job: Print "form" for a single print request. }
  lwTPPrint = ^lwTPrint;
  lwTHPrint = ^lwTPPrint;
  lwTPrint = RECORD
               iPrVersion: INTEGER; {(2) Printing software version}
               prInfo: TPrInfo; {(14) the PrInfo data associated with the
                                 current style.}
               rPaper: Rect; {(8) The paper rectangle [offset from rPage]}
               prStl: TPrStl; {(8) This print request's style.}
               prInfoPT: TPrInfo; {(14) Print Time Imaging metrics}
               prXInfo: TPrXInfo; {(16) Print-time (expanded) Print info
                                   record.}
               prJob: TPrJob; {(20) The Print Job request (82) Total of the
                               above; 120-82 = 38 bytes needed to fill 120}
               CASE INTEGER OF
                  0:
                     (printX: ARRAY [1..19] OF INTEGER);
                  1:
                     (prFlag1: TPrFlag1; {a word of flags}
                      iZoomMin: INTEGER;
                      iZoomMax: INTEGER;
                      hDocName: StringHandle; {current doc's name, nil =
                                               front window}
                      reservedAndWeMeanIt: ARRAY [1..6] OF INTEGER;
                      psFileVRefNum: INTEGER;
                      pPSFileName: Ptr;
                      lastVRefNum: INTEGER;
                      lastDirID: Longint; );
            END;
```

Figure 2—Redefined Print Record

When you print in the foreground, the LaserWriter driver uses the information in the print record to create the PostScript file. When you print in the background, the LaserWriter driver adds a resource with the information to the spool file so PrintMonitor can do the same thing. This lets this method work in both the foreground and background cases.

Be careful about picking file names to use—whatever method you use, if the file name is not unique, the LaserWriter driver will attempt to delete the existing file and cause you, as Hugo puts it, "endless grief." Simply examining the directory at spooling time isn't enough if there's any chance the file name might not be unique when PrintMonitor gets around to despooling the file. Furthermore, if you try to write over an existing file, the LaserWriter driver will not change the end-of-file position, so the result file will have the larger EOF of the new file and the old contents. This is also Bad.

Here's a summary of what we've discussed:

1. Get a print record ready to start your printing loop.
2. Change the `fPSFile` bit to tell the LaserWriter driver to create a PostScript file instead.

3. Set the vRefNum and directory ID for where you want to store the file.
4. Allocate a 64-byte pointer to store the file name; put the filename in the new memory block and put the pointer in the print record.
5. Complete your print loop as normal.
6. Zero the file name pointer in the print record as a precaution.
7. Save or otherwise manipulate the print record for future use.

Samples? Sure.

The following System 7-only sample routine PrPSFileDialog can be used in place of PrJobDialog when using this technique. Note that the routine requires a handle of type lwTHPrint.

Note: Be careful about bypassing the job dialog. The LaserWriter driver resets some other job-specific parameters (besides the file to save PostScript in) in PrJobMain. If you save a print record immediately after calling PrJobDialog and use that print record without going through the job dialog again, **all** of the job-specific parameters will be preserved (number of copies, page range, feeder choice, etc.). This may not be what you intended.

```
FUNCTION PrPSFileDialog(thePrRecHdl: lwTHPrint): BOOLEAN;

  VAR
     ourReply: StandardFileReply;
     tempFlags: INTEGER;
     tempResult: BOOLEAN;
     ourOSError: OSErr;
     tempPtr: Ptr;

  BEGIN
     PrPSFileDialog := FALSE; { being conservative }
     tempResult := PrValidate(THPrint(thePrRecHdl));

     IF NOT thePrRecHdl^^.prFlag1.fPSFile THEN
     { don't do this if there's already a file to be saved! }
        BEGIN
        StandardPutFile('Save PostScript file as','PostScript',ourReply);

        IF ourReply.sfGood THEN {Did the user accept a file to save as?}
           BEGIN
           IF ourReply.sfReplacing THEN BEGIN
              ourOSError := FSpDelete(ourReply.sfFile); { make sure this is
                                                          gone before we
                                                          start! }
              IF ourOSError <> noErr THEN AlertUser;
           END;
{OK, we're finally ready to pound the print record.  Put on your division by zero suit...}
           thePrRecHdl^^.prFlag1.fPSFile := TRUE;
           thePrRecHdl^^.psFileVRefNum := ourReply.sfFile.vRefNum;
           tempPtr := NewPtr(64); {Size of reply.fName. }
           IF tempPtr = NIL THEN AlertUser; {SERIOUSLY low on memory }
{The LaserWriter driver will Dispos of this pointer before the end of the print loop.}
           thePrRecHdl^^.pPSFileName := tempPtr;
           BlockMove(@ourReply.sfFile.name,thePrRecHdl^^.pPSFileName,
                     (LENGTH(ourReply.sfFile.name) + 1));
```

```
          thePrRecHdl^^.lastDirID := ourReply.sfFile.parID; {ID of
             user-chosen directory}
          PrPSFileDialog := TRUE;
       END {IF good THEN…}
    END {if flag was clear}
END;
```