#192: Surprises in LaserWriter 5.0 and newer

See also: The Printing Manager
LaserWriter Reference Manual

Written by: Scott “ZZ” Zimmerman April 2, 1988

This note describes some changes in the 5.0 and newer releases of the LaserWriter driver.

With the release of LaserWriter 5.0 and background printing, a few changes had to be made to the LaserWriter driver. Although these changes were transparent to most applications, some have had problems. Most of these problems were related to use of unsupported features. Here is a partial list of the changes:

No mo’ low

Because of the problems with supporting both the high-level and low-level interface in the background, the low-level interface has been all but removed. Instead of the low-level `PrCtlCall`s being executed by the device driver, the `PrCtlCall` procedure now converts the call into its high-level equivalent before execution. This way, the LaserWriter driver has a common entry point for both the low-level and high-level interfaces. Because of this conversion, the low-level calls may not be faster than using the high-level equivalents. In some cases, they may even be slower.

Are you convertible?

Whereas the conversion of the low-level calls should have been transparent, there are some assumptions made by the conversion routines. The conversion routines require a context in which to operate. The Printing Manager maintains a certain state while executing commands, and the conversion routines need access to that state to be able to perform the conversion. To provide this context, the application must have opened a document and a page. This means that the original method of using the low-level interface documented on page II-164 of *Inside Macintosh* will no longer work, as in this example:

```
PrDrvrOpen;
PrCtlCall(iPrDevCtl, lPrReset, 0, 0);
{ Send data to be printed. }
PrCtlCall(iPrDevCtl, lPrPageEnd, 0, 0);
PrDrvrClose;
```

Instead, you should use the following:

```
PrDrvrOpen;
PrCtlCall(iPrDevCtl, lPrDocOpen, 0, 0);
PrCtlCall(iPrDevCtl, lPrPageOpen, 0, 0);
{ Send data to be printed. }
PrCtlCall(iPrDevCtl, lPrPageClose, 0, 0);
PrCtlCall(iPrDevCtl, lPrDocClose, 0, 0);
PrDrvrClose;
```

This provides the Printing Manager with the context it needs to convert the calls.

Really unsupported features

Sending data to the printer in between the `PrOpenDoc`/`lPrDocOpen` and `PrOpenPage`/`lPrPageOpen` calls is not currently, and has never been supported. This data was interpreted by the LaserWriter drivers before 5.0, but this data will be ignored by the 5.0 and future LaserWriter drivers. If you would like to download some PostScript code with each document, you should use the "`PREC 103`" method described in the "Providing your own PostScript dictionary" section of the *LaserWriter Reference Manual*.

A little less control

Four of the original six `PrCtlCall`s supported by the LaserWriter driver have been discontinued due to lack of use. Also, they were difficult to support with background printing. These control calls were only supported by the LaserWriter driver, and only documented in the *LaserWriter Reference Manual*. The calls are:

- `fill`
- `hexBuf`
- `printR`
- `printF`

In addition to these calls, the `stdBuf` call has also been affected. There are two versions of the `stdBuf` call depending on the sign of the `bytes` parameter. If `bytes` is negative, the text passed to the `stdBuf` call is converted to PostScript text before being sent to the LaserWriter. This means that special PostScript characters in the text would be preceded by a PostScript escape character. Also, characters with an ASCII value greater than 128 would be converted to octal before being sent to the LaserWriter. This version of the call is no longer supported.

If the `bytes` parameter is positive, the text passed to the call will be sent as-is to the LaserWriter, and interpreted as PostScript instructions. This version of the call is still supported, but there is still one more problem. When you first open the low-level driver (via `PrDrvrOpen`) with background printing enabled, no clip region is defined. If your application then begins sending PostScript to the driver via the `stdBuf` call, all of the output will be clipped, and only a blank page will be printed.

To prevent this, you must force a clip region to be sent to the LaserWriter. The region will be sent by the driver when it receives its first drawing command. Unfortunately, the driver does not consider the `stdBuf` call to be a drawing command. To force the clip region on the printer, you can use the `iPrBitsCtl PrCtlCall` to print a small bitmap outside the printable area of the page. This will not have any effect on the document, but will fire the bottleneck routine, and cause the clip region to become defined.

Since the clip region is reinitialized at each call to `lPrPageOpen`, you should send the bitmap once at the start of each page. If any other `PrCtlCall`s precede the `stdBuf` call, you do not need to send the bitmap.

Background preparations

The "`PREC 201`" mechanism described in the "Providing your own PostScript dictionary" section of the *LaserWriter Reference Manual* will only work when background printing is disabled. This is because of difficulties finding the `PREC` resource under MultiFinder. Since the option will only work in the foreground, and since there is no way for an application to know if background printing is enabled, you should avoid using this feature.

Headin' for trouble

There is a minor bug in version 5.0 of the LaserWriter driver that will only affect applications that parse the PostScript header downloaded by the driver with each document. This header contains some PostScript comments that provide information about the current job. One of these comments is `IncludeProcSet`. This comment takes three arguments; a PostScript dictionary name, a major version number, and a minor version number. In the 4.0 version of the LaserWriter driver, the comment line looked like this:

```
%% IncludeProcSet: (Appledict md) 65 0
```

Unfortunately, in the 5.0 version of the LaserWriter driver, the last argument was removed. This caused the comment line to look like this:

```
%% IncludeProcSet (Appledict md) 66
```

Since Adobe has defined the comment to take three arguments, it is reasonable for applications that parse the comments to expect three arguments. Because of this, the 5.1 version of the LaserWriter driver will contain the correct version of the comment:

```
%% IncludeProcSet (Appledict md) 67 0
```

No go with zero

Some applications want to force a font to be downloaded to the LaserWriter without actually printing characters with the font. This can be done easily in three steps:

1. Save the current pen position.

2. Use any text drawing routine to draw a space character.

3. Move the pen back to the saved position.

Some applications have used `DrawString` with a empty string— `DrawString('')`—to force the font downloading. Although this worked in LaserWriter drivers up to 5.0, these calls will be ignored by the 5.1 and later drivers. The main reasons for this change were optimization of performance, and a reduction in the size of spool files.