



# Supporting Device Features

---

*Adobe Developer Support*

Technical Note #5117

31 March 1992

Adobe Systems Incorporated

Adobe Developer Technologies  
345 Park Avenue  
San Jose, CA 95110  
<http://partners.adobe.com/>

Copyright © 1991–1992 by Adobe Systems Incorporated. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher. Any software referred to herein is furnished under license and may only be used or copied in accordance with the terms of such license.

PostScript, the PostScript logo, Adobe, the Adobe logo, and TranScript are trademarks of Adobe Systems Incorporated which may be registered in certain jurisdictions. Apple, LaserWriter, and Macintosh are registered trademarks of Apple Computer, Inc. Microsoft and MS-DOS are registered trademarks and Windows is a trademark of Microsoft Corporation. UNIX is a registered trademark of AT&T Information Systems. Other brand or product names are the trademarks or registered trademarks of their respective holders.

*This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes and noninfringement of third party rights.*



# Contents

---

## **Supporting Device Features 5**

- 1 Introduction 5
- 2 Supporting Device Features 6
  - Hard-Coding Device Feature Support in the Application 6
  - PostScript Printer Description Files 7
- 3 Issues In Supporting Device Features 10
  - Paper Handling 10
  - Media Options 13
  - Manual Feed 14
  - Output Order 14
  - Font Support 16
  - Supporting Duplex Printing 17
  - Supporting Color Devices 19
- 4 Supporting Document Structuring Conventions 20
  - Generating Device Specific Code 20
  - Post-Processing 22

## **Appendix: Changes Since Earlier Versions 23**

## **Index 25**



# Supporting Device Features

---

## 1 Introduction

There are over 176 PostScript™ language devices currently available. One of the major advantages to the PostScript language is that it provides a device-independent method for describing how to put marks on a page. In general, any program that supports one PostScript language device will work with another PostScript language device without any modification. Therefore, supporting the PostScript language provides instant support for a large set of output devices.

But while the PostScript interpreter in each printer is the same, the physical capabilities of each device are different. The features of each printer distinguish it from others. Perhaps the printer has more memory than most or a wider selection of fonts. Maybe there are additional paper trays available or support for font cartridges. Perhaps the device is capable of sorting or printing duplex pages or has a built-in hard disk. These features are how a printer manufacturer can address the needs of a particular market segment.

If a user has chosen a specific printer with a specific feature set, he/she expects that the software will support all the special features of the printer. How does an application make the extra features of printers available to the user of the application? There are a few different techniques, as this paper will discuss.

*Note This document is undergoing a major revision. It does not address many of the device features that your application will be supporting. It does not cover most of the keywords that are new to Version 4.0 of the PostScript Printer Description File Specification. We do feel, however, that distribution of this document will still provide useful information for incorporating device feature support into your application. This document will be updated and greatly expanded.*

## 2 Supporting Device Features

In this document, the application that provides the print panel function is referred to as a print manager. This term is used to separate the task of printing a document from that of composing a document. Although, often, it is the same piece of software that converts an application's internal representation of a document to the PostScript language representation of the same document. The function of the print manager can be provided by a system-level driver, by a separate piece of software, or it might be part of an application.

It is best to functionally isolate the tasks of printing and composing as much as possible. For example, it should always be possible for the user to compose a page that cannot be printed on the user's printer. The file might eventually be printed in a completely different environment from the one available when the document was created.

At the printing stage, the decisions about page sizes and fonts have been made (by the document composition software) and the spooler or print manager must invoke these features correctly for the chosen printer. There are two things that a print manager must know: which features a printer supports, and how to invoke those features.

On a PostScript Level 1 device, device-specific operators are non-standard. There are separate operators to invoke each device feature; therefore, there might be a different set of operators on each device. Does this mean your print manager will have to have a separate driver for each different device? The answer is no. The following sections explain why.

In PostScript Level 2, the PostScript language has been revised to include a device-independent method of invoking device-dependent features. Even though each printer has a different set of features, one operator, **setpagedevice**, is used to invoke them. Therefore, having just one PostScript printer driver that generates files for Level 2 printers will be easier to achieve.

### 2.1 Hard-Coding Device Feature Support in the Application

Typically, the application printing model has one driver for each printer with hard-coded support for device features. Because one PostScript printer driver can support so many printers, it is not practical to hard-code in device feature support.

For example, one way to handle device feature invocation is to read each printer manual and hard-code each feature into the application. This would result in a large series of “if” statements (or equivalent “case” statement), that basically looks like this:

```
if (printer == "Apple LaserWriter") then
    ...
else if (printer == "TI microLaser") then
    ...
if (printer == "Linotype 300") then
    ...
if ...
```

Considering that there are over 176 PostScript language printing devices on the market, this technique would not only take a lot of time to research all the product names and the code needed to execute, it also would be out-of-date as soon as a new printer is released. Because there are many more PostScript printers in development, that would likely happen very soon.

## 2.2 PostScript Printer Description Files

PostScript printer description (PPD) files make the extra features of a printer available to software developers and, as a result, available to end users. They are intended to provide a uniform approach to using the widely varied feature sets available on PostScript printing devices. The information contained in PPD files serves both as an index of the available features and as a mechanism for invoking those features on a particular printer.

A document called *PostScript Printer Description Files Specification*, part number LPS5003, is available from Adobe Systems describing in detail the format of these files. It is intended for software developers interested in supporting these files. PPD files are the solution recommended by Adobe for supporting device-specific features and are referenced throughout this document.

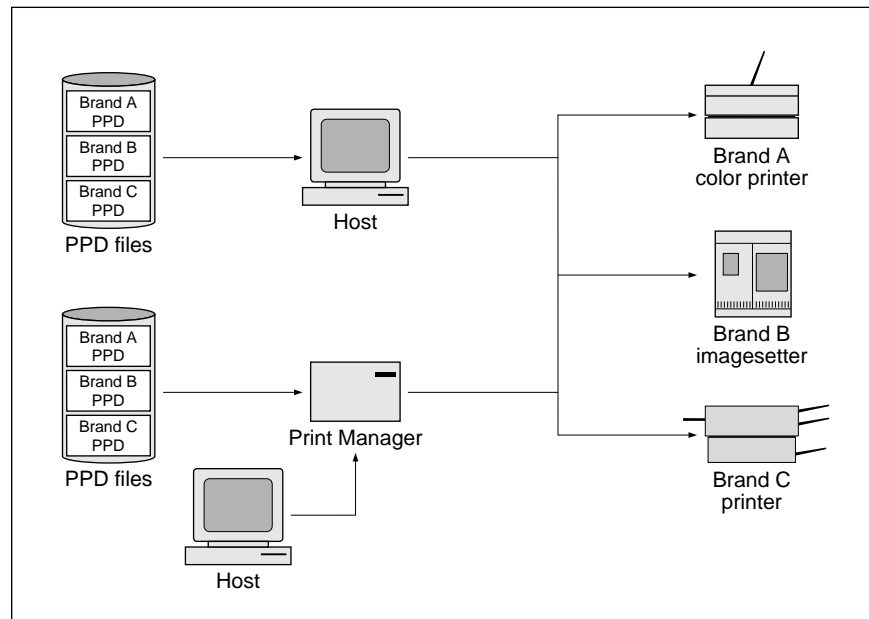
*Note* Adobe Systems has written a PPD file parser available for incorporating into your product. See Technical Note #5127, “PPD File Parser: Application Programmer’s Interface,” for more information.

### How PPD Files Work

PPD files are text files containing information about a printer in a consistent format that can be read by software. The files contain many different kinds of printer specific information that can be used by printer driver software for decision-making purposes or to invoke particular features of the printer.

There is one PPD file for each individual version of each PostScript printer, although more than one version can be combined in a single file if they are functionally identical. The PPD files reside on the host computer and/or print spooler making it easy for driver software to take advantage of them.

**Figure 1** *Where PPD files reside*



PPD files contain information including, but not limited to

- Amount of VM (memory) available
- Whether or not the device has a hard disk
- Whether or not the device outputs color
- The resident fonts, including version numbers
- The different paper sizes supported and how to invoke them
- How to set up custom paper sizes and the limitations
- Paper trays supported and how to invoke them
- Information about the imageable region for each paper size
- Whether or not the device supports manual feeding of paper
- Recommended color separation parameters
- User interface information

This information is represented as small PostScript language code segments that will invoke a particular feature or as informational keywords that can be used to make decisions about the print job.

### **Who Should Support PPD Files?**

PPD files can be supported by the computing environment or by individual applications. For example, in environments such as the Macintosh® or Microsoft Windows™, there is a system-wide driver. Drivers written by Adobe Systems for these environments support device features through PPD files.

TranScript™ (for the UNIX® environment) also uses PPD files. In MS-DOS®, where every application has its own printer driver, each application might need to support PPD files. In addition, any application on any platform that needs to access printer specific information might want to support them directly.

Among its other duties, the print manager takes input from the user through a user interface (such as a print panel or command line), extracts from the PPD file the corresponding code sequences to invoke the requested features, inserts the code sequences into the appropriate setup section of the output file, and surrounds the code sequences with the appropriate DSC comments.

### **When to Parse PPD Files**

The PPD files, as distributed by Adobe Systems and the printer manufacturers, are ASCII files that can be ported to any platform under any environment. These files can be parsed directly by the print manager software as needed. Some environments compile the information from a PPD file into a format more easily parsed by software on a given platform.

Microsoft pre-compiles the PPD files from Adobe Systems into WPD files that are then distributed with the Windows software. This pre-compiling of PPD files has the advantage that only the keywords and information that the environment (or application) actually supports are included and parsed, and they do not need to carry the extra information.

Pre-compiled PPD files can be in a binary format that is quickly read by the application that needs the data. The disadvantage of pre-compiling is that a user of a brand new printer must either wait for a company to pre-compile the PPD files and distribute them, or they must have a utility so that with a new printer, he/she can get the PPD file from the diskettes shipped with the printer, compile a PPD file for immediate use.

### 3 Issues In Supporting Device Features

There are many issues to consider when supporting any of the available device features. Some of these issues are for the document composition software, while others are for print managers. The following sections present these issues for some of the categories of device features.

#### 3.1 Paper Handling

There are many different standard paper sizes. These include

A3	B4	Ledger	Quarto
A4	B5	Legal	Tabloid
A5	Executive	Letter	Statement
B3	Folio	Postcard	10 x 14

In addition, some devices, such as roll-fed imagesetters, can handle custom paper sizes because the paper or film will be cut to the desired size after it has been imaged.

#### User Interface Considerations

There are a number of different methods an application can choose to display paper size choices to the user at composition time.

- *The list can be fixed and hard-coded into the application.* This method provides the user with the basic paper sizes, and would probably work for the majority of cases where a user wants to write a standard business letter, for example. However, the limitations are that the user does not have the flexibility to design a page that is not hard-coded into the application. And a user with a printer that can print to “non-standard” paper sizes is unable to take advantage of that from an application that has a fixed list of “standard” paper sizes.
- *The list can be generated from a PPD file.* This allows the user to design a document to print on non-standard paper, but only if they had the PPD file for a printer that supported the page size they wanted. Deriving the list from any one particular device limits the flexibility to print on any device once the document is designed and created. If the user has access to a wide range of PPD files, however, they can theoretically compose any page size supported by some device.
- *The application could provide a mechanism for the user to design any page size (standard or not).* Additionally, the application could provide a menu of common sizes from which the user can choose. This has the benefit of full flexibility for the user, as well as ease of use for choosing standard page sizes.

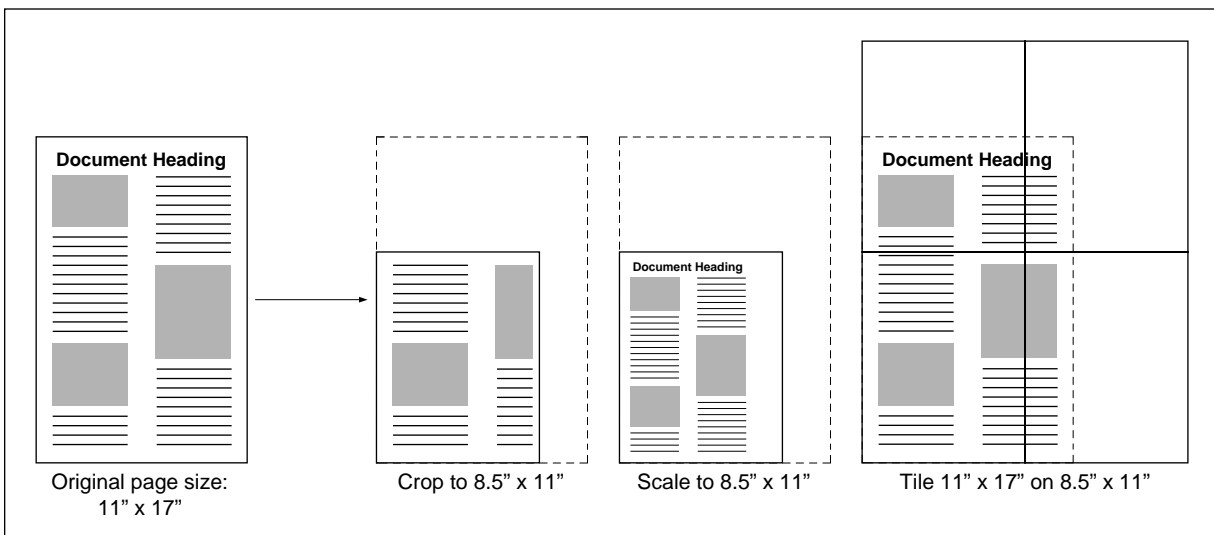
## Printing Considerations

For the user to choose a paper size on which to print the desired document separately from the choice used to create the document allows the user maximum flexibility. This allows for a distinction between the document's desired page size and the paper size on which the document is to be printed. At print time, the list of paper sizes is derived from the PPD file, which lists available paper sizes the target printer can support.

For example, a user can design a page that is 11 inches by 17 inches and will eventually be printed on corresponding paper. But they wish to proof the document on a printer that does not support that paper size. The print manager can have a strategy for printing a document on a different paper size than the one for which it was designed. The following are several options:

- *crop* the page design so that whatever can fit on the paper is printed and the rest of the document does not appear
- *scale* the image so it can fit on the available paper size
- *tile* the image across several pieces of the available paper size
- produce an *error* and not print the document

**Figure 2** *Printing when the desired paper size is not available*



The print manager might want to provide the user the option of selecting what to do.

## Level 1 Versus Level 2

These print strategies are easy to implement in Level 1 using the **scale** and **translate** operators.

To crop the page, the application does not need to do anything. Just send the entire page description to the printer, and the PostScript interpreter will crop the image to the available imageable area on the page.

To scale the page to a smaller size, calculate the scale factor necessary. You will have to decide whether or not to keep the aspect ratio the same as the original.

```
/scaletofit {                                % llx lly urx ury  scaletofit -
  /ury ed /urx ed /lly ed /llx ed  % box to fit into
  newpath clippath pathbbox
  5 sub /iury ed 5 sub /iurx ed 5 add /illy ed 5 add /illx ed
  iurx illx add 2 div iury illy add 2 div translate
  iurx illy sub iury illy sub ury lly sub div exch urx
    lls sub div
  2 copy gt {exch} if pop dup scale
  urx llx add -2 div ury lly add -2 div
  translate newpath} bind def
```

To tile the image, print the page several times, translating the user coordinates so a different section of the image is printed on each page.

It is even easier to provide this functionality in Level 2. The operator **setpagedevice** can invoke various policies that implement a backup strategy for what to do if a requested feature is not supported. There are several built-in policies that can be chosen if the requested page size is not supported.

For example, the following code requests 11 x 17 inch paper, but also sets **Policies** to revert to the nearest page size and scale the image to fit the new page size.

```
<< /PageSize [792 1224]
  /Policies << /PageSize 3>> >> setpagedevice
```

See section 4.11 of the *PostScript Language Reference Manual, Second Edition* for more information on **setpagedevice** and policies.

## PPD Keywords

The appropriate keys to reference in PPD files for media handling are:

- \*PageSize
- \*PageRegion
- \*PaperDimension
- \*ImageableArea
- \*CustomPageSize
- \*ParamCustomPageSize
- \*LanguageLevel

See the *PostScript Printer Description Files Specification* for more information.

## 3.2 Media Options

Many printing devices have multiple input slots that a user can load with different sizes of paper, different kinds of paper (for example, colored, weighted, letterhead, and so on.), or envelopes. They might want to have letterhead in a specific tray, so when they print business letters they can print directly on letterhead, while all other jobs print on regular white paper. Or they might want to send a job that contains requests for several different types of media in the same job. For example, they might want to print the first page of a business letter on letterhead, the second page on second page letterhead, and the third page on an envelope.

These kinds of special request jobs can also be achieved using manual feed, although this requires the user to be at the printer at the time the job is sent. In a networked environment, a user cannot always determine if the next job is his/her job, and, therefore, whether it is time to feed special paper. In addition, it is cumbersome to manually feed a job.

### User Interface Considerations

The application should present a user interface that allows the user to choose the input tray. This might be a simple design that allows one tray choice for the entire job, a more advanced option that allows the first page to be printed from one tray and the rest of the job from another tray, or one that allows each page to be printed from a chosen tray.

## PPD Keywords

See the \*ImageableArea and \*InputSlot keywords in the PPD Specification.

### 3.3 Manual Feed

Sometimes, the user might want to feed the paper to the printer manually, because he/she wants to print on unusual paper but does not want to keep it loaded in any of the input trays, or the user might have a printer that does not have multiple input trays.

#### User Interface Considerations

A simple user interface offering a *Manual Feed* boolean or check box is sufficient to support manual feed, or it might be listed as a special case of choosing a tray option. In either case, it would be useful to only offer the user the option if the target printer supports manual feed.

#### Printing Considerations

The code for setting the manual feed option on the printer has to be executed before any drawing occurs on a page. This is because in Level 2, **setpagedevice**, which is the operator used to turn on manual feed, initializes the graphics state.

#### Level 1 Versus Level 2

If PPD files are used to invoke manual feed, there should be no Level 1 versus Level 2 issues to be concerned about.

#### PPD Keywords

A PPD file can specify whether or not a printer has a manual feed option, and how to turn it on or off. See the following keys in the PPD Specification for more information:

- \*ManualFeed
- \*PageRegion

### 3.4 Output Order

Different kinds of printer designs have different paper paths that dictate whether the printed paper ends up face-up or face-down in the output bin. While some printers have multiple output bins, with possibly different ordering, other printers only have one output bin, with a fixed ordering.

A print manager can be user-friendly by determining in which order the pages will appear when they come out of a printer, and order the print job data so the pages will always come out ordered from Page<sub>1</sub> to Page<sub>n</sub>. This will save the user from having to physically reverse the order of the pages after they have been printed.

For example, if the printer places the printed pages face-up in the output bin, sending the pages in reverse order (the last page first) will allow the user to pick up the job without doing anything with it before stapling it or putting it in a copier. Similarly, if the printer places the pages face-down, then the print job can be sent with the order of the pages from Page<sub>1</sub> to Page<sub>n</sub>.

### **User Interface Considerations**

The application might want to ask the user whether to print the document from Page<sub>1</sub> to Page<sub>n</sub> or from Page<sub>n</sub> to Page<sub>1</sub>. The application could allow the user to specify whether they want the pages face up or face down, or the application could make this decision for the user.

### **Printing Considerations**

To be able to specify the pages in any order, the PostScript language description of each page must be context-independent from the others. That is, any page cannot rely on whether certain fonts have been downloaded on a previous page, or make assumptions about the graphics state as a result of some earlier page being executed.

### **Level 1 Versus Level 2**

If PPD files are used to invoke manual feed, there should be no Level 1 versus Level 2 issues to be concerned about.

### **PPD Keywords**

The PPD file can be used to determine in which order the printer puts the pages in each output bin, and also whether that can be changed, because some printers have multiple output bins that can be selected through software.

For more information, see the following keys:

- \*DefaultOutputOrder
- \*OutputOrder
- \*DefaultOutputBin
- \*OutputBin

### 3.5 Font Support

There are both document composition issues and print manager issues that must be considered with regard to proper font support. The document composition software must provide a user interface for allowing font choices. This list should not be hard-coded into an application.

For example, there are applications that only allow the user to choose from the standard 13 list of fonts that all PostScript printers ship with. Other applications hard code the list of standard 35 fonts that ship in the Apple® LaserWriter® and many other printers. But there are many printers (at least 26) that ship with more fonts or a different set of fonts. The users might have chosen them precisely for their additional fonts only to find that an application they are using does not allow them to access the extra fonts.

In addition, there are over 10,000 “after-market fonts” that users can buy. Again, users have spent money to purchase these extra fonts—and they expect their applications to support them. If their printer has a disk, the user can download the fonts permanently to the disk. Otherwise, they expect the system to download the font on an “as-needed” basis with each document that is to be printed.

For the fonts to be used in your application, it will need to be aware of font-specific information such as characters available, character widths, and bounding box data. This information can be derived from the appropriate Adobe™ Fonts Metrics (AFM) files. See Technical Note #5075, “Supporting Fonts in PostScript Environments,” for more information.

#### **User Interface Considerations**

Each application or system needs to have a way for users to add fonts to their system and application menus so that the fonts are usable from your application. A separate installer application can be shipped with the system or application.

#### **Printing Considerations**

The print manager must have a strategy for determining which fonts are required to print a job, which fonts are already on the printer, and if it is possible to download the remaining fonts with the job to the printer (whether there is enough memory in the printer to handle the fonts).

If there is bidirectional communication with the printer, then a special query job can be sent to the printer to determine which fonts it has. For more information about query jobs, see the document structuring conventions specification (Appendix G in the *PostScript Language Reference Manual, Second Edition*).

There are two queries supplied in the PPD file. The code in the `*?FontList` keyword lists all available fonts on the device. The code in the `*?FontQuery` keyword checks for specified fonts. The query code supplied in the PPD file should be used instead of writing your own, because it is written appropriately for the specific device to check all possible font storage locations that the printer has, including ROM, disk, and cartridges.

If the print manager cannot establish bidirectional communication with the printer, then the next best strategy is to check the PPD file to see which fonts are shipped with the printer. This can be determined by using the `*Font` and `*ExtraFont` keywords.

Once it has been determined that a font must be downloaded, then the print manager needs to know how much memory is available in the printer to help determine the downloaded strategy (for example, how many fonts can be downloaded at one time, and so on). If there is bidirectional communication, the `vmstatus` operator will give appropriate information. Otherwise, check the `*FreeVM` keyword in the PPD file for an estimate.

### **Level 1 Versus Level 2**

If PPD files are used to invoke manual feed, there should be no Level 1 versus Level 2 issues to be concerned about.

### **PPD Keywords**

Use `*Font` and `*ExtraFont` for information about fonts that reside on the printer. Use `*FreeVM` to determine how much memory is available to help determine your font downloading strategy. Use `*?FontList` and `*?FontQuery` to determine which fonts have already been downloaded (when you have bidirectional communication with the printer).

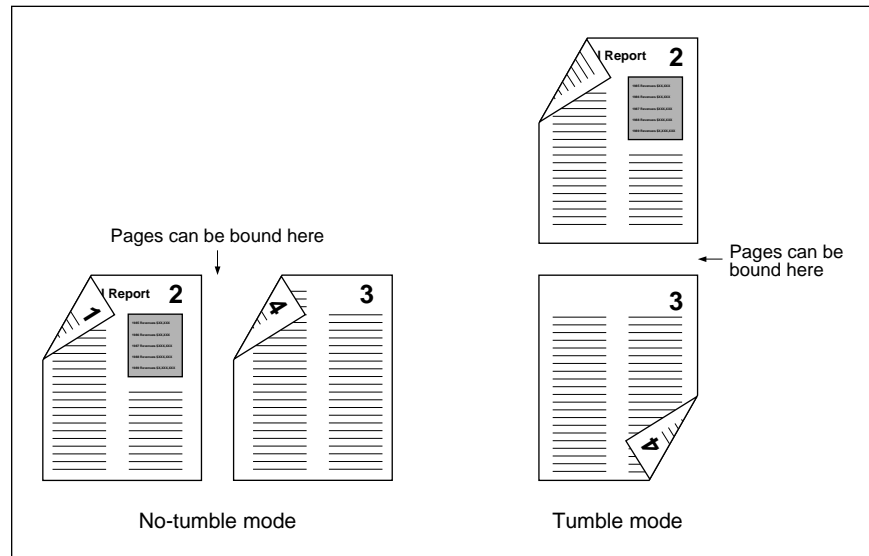
## **3.6 Supporting Duplex Printing**

Duplex printing is printing on both sides of the same sheet of paper. Several PostScript printers currently support duplex printing, with more devices expected soon.

Printing duplex should be considered a print manager function. The user should be allowed a choice at print-time to print any document (regardless of how it was formatted) in duplex mode.

There are two modes of duplex printing: tumble and no-tumble. When printing in the no-tumble duplex mode, the output is suitable for binding on the left or right sides of a sheet of paper. Printing in the tumble duplex mode, produces output that is suitable for binding at the top or bottom of the paper.

**Figure 3** *No-tumble versus tumble*



### **User Interface Considerations**

You might want to add extra features to the composition software because there could be different formatting needs for documents that will eventually be printed in duplex mode. This might require allowing a distinct document layout for even and odd (or left and right) pages. For example, users might want a wider margin on the side of the pages where they will be bound. This compensates for the fact that the binding process decreases the margins.

### **Level 1 Versus Level 2**

If PPD files are used to invoke manual feed, there should be no Level 1 versus Level 2 issues to be concerned about.

### **PPD Keywords**

To determine if a device is capable of printing in duplex mode, see the \*Duplex key in the PPD Specification.

### 3.7 Supporting Color Devices

There are currently 18 color PostScript output devices. As the color printing technology becomes less expensive to make and use, more and more users will have access to color output devices, and they will be more demanding of applications to efficiently support color.

The PostScript language has always had support for specifying color values in both the RGB (Red-Green-Blue) and HSB (Hue-Saturation-Brightness) color models. When a job specifies color values in either of those color models the PostScript interpreter does appropriate halftoning to achieve reasonable results.

When the first color PostScript printers were implemented, support for the CMYK (Cyan-Magenta-Yellow-Black) color model was added to the language. Because this was an extension to the language, the CMYK operators are, by definition, not in all Level 1 interpreters. Therefore, an application generating a color PostScript language job either needs to know if the target device has those operators, or it must provide emulations for the operators, so the job will not fail. This can be determined by checking the `*Extensions` key in the PPD file. All Level 2 devices contain those operators, whether or not the device can actually output color. The `*LanguageLevel` key in the PPD file gives this information about a specific printer.

In addition `*DefaultColorModel` tells you whether the device is, by default, a CMYK, RGB, or grayscale device. This enables an application to convert data to the appropriate color model before sending the job, if desired.

Another strategy to consider is to downsample CMYK/RGB image data to black image data when sending the print job to a black and white device. This reduces transmission time because the file size is significantly smaller. However, the disadvantage is that if there is any chance the file will be captured to disk or intercepted by a spooler, it will be to grayscale data instead of the original color data. This technique should only be used if you are certain the job will go to the targeted device. The key `*ColorDevice` in a PPD file contains the information about whether a device actually produces color output or not.

#### User Interface Considerations

A user interface issue that a print manager might want to support is color breakout. This is the ability to target individual pages of a document to be sent to a color printer while the remaining pages are sent to a black and white device. This is useful to a user who has a report to print, most of which is black text but has a color chart in the middle of the document. Instead of wasting color resources to print all the black and white pages on a color printer, they can be sent to a black and white device instead.

## Level 1 Versus Level 2

If PPD files are used to invoke manual feed, there should be no Level 1 versus Level 2 issues to be concerned about.

## PPD Keywords

See the following keywords in the PPD Specification:

- \*Extensions
- \*LanguageLevel
- \*ColorDevice
- \*DefaultColorModel

## 4 Supporting Document Structuring Conventions

### 4.1 Generating Device Specific Code

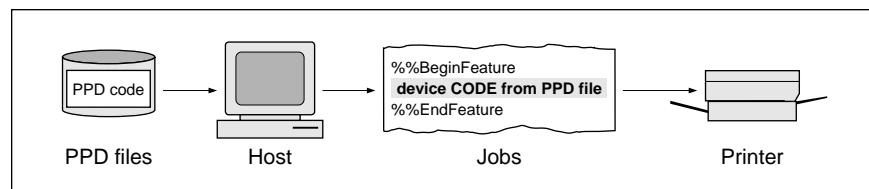
When the print manager includes the device-specific code for the features requested by the user, it should surround these feature requests with document structuring conventions (DSC) comments, for possible later parsing by other applications, spoolers, or print managers.

The example below shows a PostScript language output file that describes a small document. In this example, the output file does not yet contain DSC comments or device-specific code. Throughout this section, this output file will grow as DSC comments and device-specific code are added:

```
/sp /showpage load def
100 100 translate
20 50 moveto
20 100 lineto
stroke
sp
```

In the next example, assume that the user requested letter-size paper using a user interface. The print manager extracts from the PPD file the device-specific code to invoke letter-size paper, inserts the code into the output file, inserts the appropriate DSC comments, and sends the output file to the output device.

**Figure 4** *Supporting the DSC*



The following is the sample with DSC comments and device-specific code added. The code sequence extracted from a PPD file is in boldface.

```

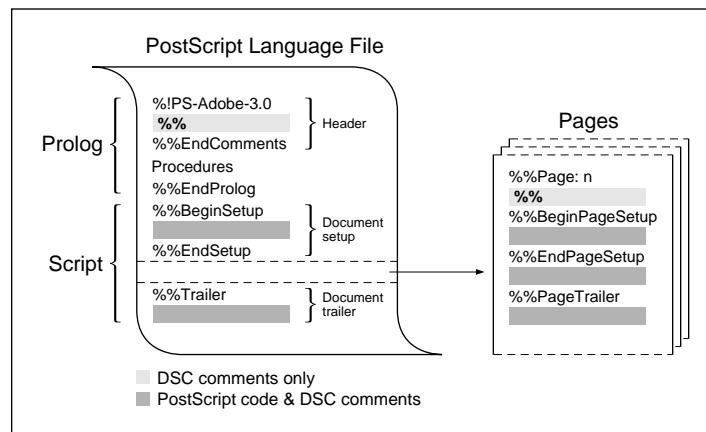
%!PS-Adobe-3.0
%%Title: test.ps
%%EndComments
/sp /showpage load def
%%EndProlog
%%BeginSetup
%%BeginFeature: *PageSize Letter
statusdict begin lettertray end
%%EndFeature
%%EndSetup
%%Page: one 1
100 100 translate
20 50 moveto
20 100 lineto
stroke
sp
%%Trailer

```

When the output file is sent to the output device, the interpreter ignores the comments and executes the PostScript language commands, including the code sequence that sets up the letter-size input tray.

There are several different sections of a PostScript language job. Two major sections of the document are the prolog and the script. The prolog is typically a set of procedure definitions appropriate for the set of operations a document composition system needs, and the script is the software-generated program that represents a particular document. In general, device setup code should be executed from within either the document setup section or the page setup section, both of which reside in the script.

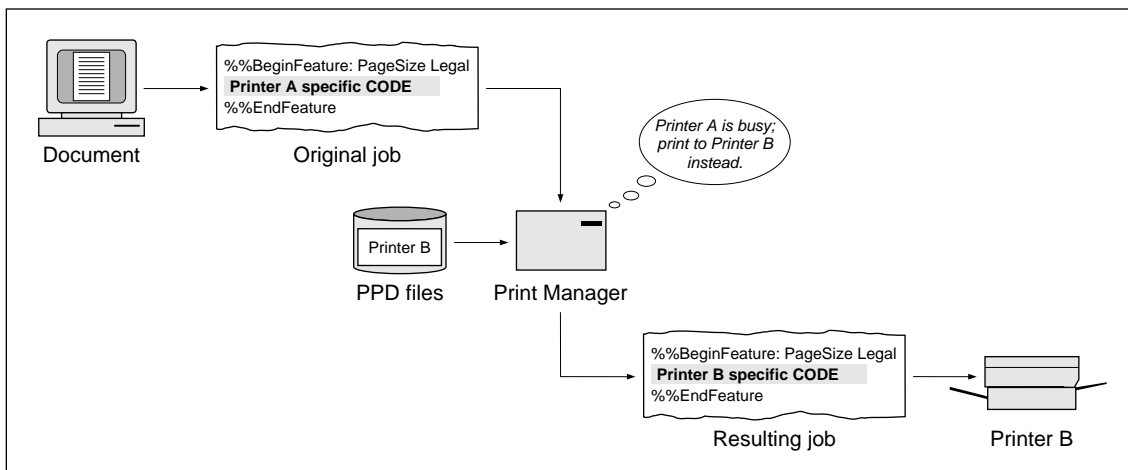
**Figure 5** *Anatomy of a PostScript language job*



## 4.2 Post-Processing

In some environments, there might be a post-processor, such as a spooler that also acts as a print manager. In this context, the requested device might be unavailable, and the print manager/spooler might need to redirect an output file from one device to another. If an output file is to be redirected, the print manager parses the DSC comments in the output file, and strips out the original device-specific code. It then parses the PPD file of the newly selected device, extracts from the new PPD file the device-specific code requested by the DSC comments, inserts the device-specific code from the new PPD file into the output file, and sends the output file to the new device.

Figure 6 *Print spooler intervenes*



Here is the example file as it is sent to the new device. (Note that the device-specific code is different from the code in section 4.1.)

```

%!PS-Adobe-3.0
%%Title: test.ps
%%EndComments
/sp /showpage load def
%%EndProlog
%%BeginSetup
%%BeginFeature: *PageSize Letter
  1 dict begin /PageSize [612 792] currentdict end setpagedevice
%%EndFeature
%%EndSetup
%%Page: one 1
100 100 translate
20 50 moveto
20 100 lineto
stroke
sp
%%Trailer

```



# Appendix: Changes Since Earlier Versions

---

## Changes since August 9, 1991 version

- Document was reformatted in the new document layout and minor editorial changes were made.



# Index

---

## C

CMYK color model 19  
color devices 19–20  
    user interface 19  
\*ColorDevice 19

## D

\*DefaultColorModel 19  
device features  
    font support 16–17  
        PPD keywords 17  
        printing 16  
        user interface 16  
    hard coding 6  
    issues in supporting 10–20  
    supporting 5–22  
        composing 6  
        print manager 6  
        printing 6  
document structuring conventions  
    20–22  
    device-specific code 20  
    post-processor 22  
    print spooler 22  
downsample 19  
DSC. *See* document structuring  
    conventions  
\*Duplex 18  
duplex printing 17–18  
    no-tumble 17  
    tumble 17  
    user interface 18

## E

\*Extensions 19  
\*ExtraFont 17

## F

\*Font 17  
\*FreeVM 17

## L

\*LanguageLevel 19

## M

manual feed 14  
media options 13  
    PPD keywords 13  
    user interface 13

## O

output order 14

## P

paper handling 10–13  
    Level 1 versus Level 2 12  
    PPD keywords 13  
    printing 11  
    user interface 10  
paper sizes 10  
Policies 12  
PPD (PostScript Printer Description)  
    files 7–9  
    contents 8  
    parsing 9  
    supporting 9  
prolog **21**

## S

**scale** 12  
script **21**

**setpagedevice** 6, 12

## **T**

**translate** 12

## **V**

**vmstatus** 17