PPR, a Print Spooler for PostScript

PPR, a Print Spooler for PostScript

David Chappell

7 March 2003

Table of Contents

Abstract

This document is intended to describe PPR's features and theory of operation more deaply than *Installing and Using PPR* does. Unfortunately, this document has never been finished.

Introduction to PPR

PPR is designed to operate printers which use page description languages compatible with the PostScript language defined by Adobe Systems. It is true that a PostScript printer can generally be made to work with an ordinary print spooler. Enhancements, generally in the form of special filters or interface programs, are available for both the System V LP spooler and the Berkeley LPR spooler. However, these are generally rather limited adaptations written to fulfill a pressing need, such as the need to print banner pages. In order to fully exploit the more sophisticated features of PostScript and the Adobe Document Structuring Convention, it seemed best to start from scratch, writing a print spooler designed with modern PostScript laser printers in mind.

It is one of the goals of this project to produce a print spooler which can accept print jobs from many different types of computer systems simultaneously. This includes logged−in Unix users, MS−DOS and MS−Windows users, Unix users on remote systems through the LPR/LPD protocol, and Apple Macintosh computers.

The spooler is designed to provide services such as font and resource caching which should reduce the time a Macintosh computer requires to generate and transmit the print file.

On MS−DOS systems, many application programs are not able to generate PostScript themselves. For this reason, PPR has an automatic conversion system which attempts to determine the format of an incoming document and convert it to PostScript, if necessary. Filters to convert simple line printer files or files for popular dot matrix printers are included with PPR.

PPR will read Adobe PostScript Printer Description (PPD) files in order to obtain information about the printers under its control. PPR will replace specially marked code fragments in the document with the code fragments from the PPD file. Apple's LaserWriter 8 driver properly marks the relevent code fragments. If one uses Apple's LaserWriter driver version 8 or later it is possible for a document to print correctly even if it is printed on a different printer from that originally intended and the job uses special commands such as those for bin selection. It is also possible for an application to request that PPR insert code fragments from the PPD file in order to invoke specific features of the printer, such as duplexing or legal size paper.

PPR provides a mechanism whereby the operator specifies the size and type of paper in each of a printer's bins. When printing, users will request the desired paper. Print jobs will be held back until the operator changes the paper and indicates to the spooler that he has done so. If a job is being held back for this reason, the names of the types of media required will be indicated in the queue listing. When the document is printed, PPR will insert code to select the input tray which holds the desired medium.

Where the printer and its interface support it, PPR will capture PostScript error messages resulting from the execution of a job and put them in a log file which may be printed on the banner or trailer page or later viewed by the user. If a PostScript error occurs during the printing of a job, that job is held in the queue so that it may be examined to determine the reason for the failure.

PPR can detect printer malfunctions and deal `intelligently' with them. If problems persist, it will inform an operator.

It is desireable that PPR be able to reliably identify the users on various computer systems, employing passwords when necessary, so that users can be made to pay for their use of the printers. Unfortunately, the necessary client−side software is not always available.

PPR can maintain a list of user accounts, each of which contains money. When PPR prints a job for a user on a printer which has a per page charge, PPR computes the number of sheets of paper used and deducts the corresponding amount of money from the user's account. The system administrator may set a credit limit for each user. When the credit limit is exceeded, PPR will refuse to accept more jobs for the user.

PPR can convert a document for N−Up printing, thereby saving paper. It can also convert many documents to signatures or booklets.

A Brief Note On Terminology

This document describes a complete system which accepts print jobs, places them in a queue, and prints them as soon as it becomes possible. This printing system is called PPR. Unix users submit jobs to this system by means of a program called **ppr**. Unfortunately, in discussing this system it is possible to confuse the two. This document attempts to ease that confusion by refering to the two in slightly different ways. When the spooling system as a whole is meant, it is refered to simply as PPR, in all upper case. When the program users employ to submit jobs to PPR is meant, it is refered to as **ppr** in all lower case and bold face. Further, the program for submitting jobs will frequently be refered to as, "the program **ppr**."

Core PPR Components and How They Work Together

This section describes the subprograms which comprise the core of PPR and the rôle of each.

How Jobs Enter the Queue

All jobs which enter the queue pass through the program called **ppr**. This is true even of those which pass first through the Macintosh print server, the LPR compatible server, or Samba. (These additional server components will be described later.) The program **ppr** reads a PostScript file and splits it into 3 files. These three files are placed in a spool directory and a fourth file, called the queue file, is created which describes the job and the options the user has selected. Once all the files are ready, **ppr** sends a message to the print daemon, which is called, **pprd**, telling it that a new job is in the queue. This process is illustrated in figure Figure 1.

Figure 1. A job Enters the Queue

How Jobs are Printed

When **pprd** determines that the time has come to print a file on a printer, it invokes a program called **pprdrv** which reads the 3 job files and reasembles them into a PostScript file. While doing so it will consult information read from the printer configuration file and the queue file and a PostScript Printer Description (PPD) file. It may also include fonts and other resources in the data stream. This process is illustrated in figure Figure 2. (The inclusion of fonts and other resources has been ommited from the figure for the sake of simplicity.)

Figure 2. PPR Prints a Job

Figure 3. Multiple Instances of pprdrv for Multiple Printers

The program **pprdrv** sends the file to the printer by invoking an interface program. It is the responsibility of the interface program to accept the print job on standard input, send it the printer, if possible receive any error messages, and pass them back to **pprdrv** by writing them to stdout.

Notice in figure Figure 3 that printing is taking place on PRINTER 1 and PRINTER 3.

PRINTER 2 is idle, so the processes necessary to print on it are shown with dotted line boxes, indicating that they do not exist at present.

Once the job has been printed, the interfaces program exits, after which **pprdrv** exits. The main spooling daemon, **pprd** detects this and removes the job from the queue and attempts to inform the user that the job is done.

Job Errors

When exiting, **pprdrv** will indicate to **pprd** that printing was successful or that it was not. If **pprdrv** indicates that a problem occured which was caused by the particular job being printed, a PostScript error for instance, then the job is placed in the "arrested" state, the user who submitted it is informed, and **pprd** looks for another job to print on that printer. The arrested state is a special case of the "held" state. An arrested job remains in that state until it is canceled or released.

When a job is arrested, the user who submitted it is informed, however if the arrested job is subsequently deleted by an operator, the user will not be informed as he would be if a normal job of his were canceled.

Printer Errors

If **pprdrv** indicates that the problem is with the printer, then the job is returned to the queue, the printer is placed in fault mode, and **pprd** looks for another printer on which to print the job. When **pprdrv** discovers a problem with a printer, it indicates to **pprd** whether or not it is a problem from which the printer is likely to recover without the intervention of the system operator. Problems from which it is likely to recover on its own include those which are transient or which an ordinary user might correct, such as printer−turned−off and out−of−paper. If the problem is one from which the printer may recover, then it is placed in fault−retry mode; if it is not expected to recover, it is placed in fault−no−retry mode.

In fault−retry mode, **pprd** tries to restart the printer at intervals. The delay before the retry is 10 seconds the first time and increases by 10 seconds after each failure until the interval reaches about 10 minutes, at which point it does not continue to increase. In fault−no−retry mode, the printer is not restarted until a system operator restarts it. An operator may restart a printer with the command **ppop start printer**.

When **pprdrv** detects a printer fault, it writes a description of it to a file in the `alerts/` directory, a file which is named after the printer. At intervals, **pprd** may send the contents of this file to a system operator. The interval is defined in the printer configuration file. The interval is expressed as 'every `N` faults.' If the interval is set to `1`, the alerts log is sent after each printer fault; if the interval is set to `5`, the alerts log is sent after every 5th printer fault. If the interval is set to `0`, the alerts are never sent.

If the alert interval is negative, it is treated differently. First of all, the absolute value is used as the alert interval. Second, the alerts are only sent to the system operator once. Third, a notice is sent to the system operator if the printer sucessfully prints a job after incurring at least abs(`N`) alerts.

Each time a fault occurs, the alert message is appended to the alert log. The exception is that if the previous alert occured more than one hour previously, the alert log is truncated to zero length and a new one started. Whether or not the alerts have been dispatched to an operator, the alert log may be read at any time with the command **ppop alerts** ***`printer`***.

It is in the printer configuration file that the alert interval is specified, as well as the method by which alerts are to be dispatched and the address to send them to. A typical alert dispatch method (indeed, the only one supported at this time) is `mail`. This means that the alerts are sent by electronic mail to the indicated address. The alert parameters may be set with the **ppad alerts** command.

Printers and Groups of Printers

When printing a document with PPR it is necessary to specify a set of one more more printer on which it may be printed. If you specify the name of a printer, PPR will print the document on that printer if it determines that the printer has the necessary capabilities. If you indicate a group of printers to print the document on, PPR will print the document on the first available printer with the necessary capabilities. Both printers and groups are identified by name. A printer or group name may be up to 16 characters long and may contain any printable characters except, slash (`/`). It also must not begin with tilde (`~`) or period (`.`).

Each printer is described by a printer configuration file in the directory `/etc/ppr/conf/printers/`. Each group is defined by a group configuration file in the directory `/etc/ppr/conf/groups`. The format of a printer configuration file is described in Appendix ??? and the format of a group configuation file is described in Appendix ???. The desired printer or group may be specified with the `−d` switch to the **ppr** command. If the `−d` switch is ommited, **ppr** takes the printer or group name from the environment variable `PPRDEST`. If `PPRDEST` is not defined, **ppr** attempts to send the job to a printer or group called ``default''.

Accepting Jobs From a Network

Users on the machine running PPR can submit jobs simply by running the program **ppr**. But, when the user is on a remote machine connected by a network to the machine with the PPR spooler, things get a little more complicated.

Accepting network jobs requires a daemon process which waits new connexions. When it receives a connexion, it must launch a server process. The server process will accept the print job and run **ppr** to submit it.

Often, the user on the remote computer will not have an account on the computer that is running PPR. Therefor, the server process must run **ppr** under some system user ID. This system ID may be `ppr` or any user listed in `/etc/ppr/acl/pprprox.allow`. The system user will invoke **ppr** with switches such as `-f`, `-X`, and `--charge-to`. These switches will indicate that the system user is submitting the job on behalf of a user on a remote machine.

The `-f` switch sets the name of the user who submitted the job. This name will appear in queue listings and on banner and trailer pages.

The `-X` switch is used to provide a string which uniquely identifies the user on the remote machine. Generally, this string will be the user's name on the remote machine, followed by an `@`, followed by the network address of the remote machine.

The `--charge-to` switch is useful when using the charge accounts which can be set up with **ppuser**. If the argument of the `-f` switch is the user's real name, the `--charge-to` switch can be used to specify a more cryptic account name to which the cost of printing printing should be charged.

Accepting Jobs From Macintoshes

PPR provides a daemon called **papsrv** which acts as a Macintosh print server. It accepts connexions from Macintosh clients, answers queries using information in the PPD files, and accepts jobs. Whenever **papsrv** accepts a connexion from a Macintosh, it forks a copy of itself to service the connexion. When the child daemon receives a print job, it launches **ppr** on the end of a pipe to process it.

Accepting Jobs Using the LPD Protocol

PPR provides a daemon called **lprsrv** which can be launched by the internet super daemon, **inetd**. The daemon **lprsrv** implements the Berkeley LPD protocol.

When a print job is received, **lprsrv** checks to see if the destination specified matches the name of any PPR printer or group. If it does, **ppr** is invoked to accept the job. If not, the job is fed to the system's default spooler through the **lp** or **lpr** command. There is a **lprsrv** configuration file, `/etc/ppr/lprsrv.conf`, but it is used for access control, not for choosing which queues will be shared. All PPR printers as well as all the printers belonging to the normal system spooler are automatically made available.

However, unless **inetd** invokes **lprsrv** with the `-p` (permissive) switch, it will only accept jobs from machines listed in `/etc/hosts.equiv` or `/etc/hosts.lpd` which are also not listed in `/etc/hosts.lpd_deny`. (It is also ok to list a whole domain in these files. Any name begining with a period is interpreted as a domain name.)

Accepting Jobs From a LAN Manager For Unix Server

AT&T's LAN Manager for Unix can invoke a print processor script when a job is received for a certain printer. PPR provides a print processor script which submits the job using the **ppr** command. If the directory `/var/opt/lanman/customs` exists when PPR is installed, then the script is installed as `/var/opt/lanman/customs/ppr`. While it is possible to prepare several versions of this script to invoke **ppr** with various options, it will probably prove more convenient to use the **ppad switchset** or **ppad group switchset** command (depending on whether it is a printer of a group) to store the desired switches in the printer or group's configuration file.

Accepting Jobs From a Samba Server

The free Unix file and print server Samba has a very flexible printing system. The print command for any given print queue can be specified in Samba's configuration file. The Samba server sends jobs to PPR by writing the job to a temporary file and then invoking **ppr**. The `-U` switch should be used to tell **ppr** to delete the temporary file once it is done reading it. It is advisable to append `&` to the print command because for very large files which must be filtered, the client will time out before **ppr** exits. Sample Samba print commands are given in the ppr2samba(8) man page.

Identifying Users

It is often necessary to identify the user who submitted a job. In the case of a printer which is shared by may users, a banner page giving the user's name may be required. It is also helpful in queue listings to identify the submitting user of each job. When it is desired to restrict printing privledges or to charge for printing, passwords may be in order.

Because PPR accepts print jobs from many different kinds of computer systems it employs several techniques for determining and verifying the identity of users. Basically, PPR has two user validation modes. The first is called system validation mode. In system validation mode, PPR assumes that since the user gained access to the system and managed to execute **ppr** he must be authorized. In system validation mode security efforts are concentrated on preventing the user from disguising his identity to avoid printing charges. The other validation mode is authorization code validation mode. In that mode, an authorization code (password) is embedded in the job as a PostScript comment or communicated using a special protocol. Such a password scheme is advisable when the jobs are arriving over an unsecured channel, such as an AppleTalk or LAN Manager network.

The means by which the user name is determined depends on the validation mode. In system validation mode, it is the user name of the user who invoked **ppr**, unless the user is `ppr`, `root`, or is listed in `/etc/ppr/acl/pprprox.allow`. These are priveledged users who may accept jobs over the network on behalf of remote users.

When a privledged user is submitting a job on behalf of another, the name may be supplied with **ppr**'s `-f` switch. A privledged user may also use the `-R for` switch to indicate that `%%For:` lines in the input file are to be scanned for the user name. Of course, if the privledged user employs none of these options then the job will be identified as belonging to the privledged user who submitted it.

The name that the priveledged user supplies with the `-f` switch will often be a 'real name' such as "Joseph Andrews". If printer charge accounts use a shorter name such as `jandrews` then the `--charge-to` option will be used to indicate that name. In that case, the privledged user might invoke **ppr** like this:

```
$ ppr -d thatprn -f "Joseph Andrews" --charge-to jandrews
```

There must also be some mechanism to make sure that a user can't delete another user's jobs. For that reason it is necessary to unambiguously identify each remote user from which a privledged user accepts jobs. The privledged user uses the `-X` switch to inform **ppr** of this identification. So, the command above would become:

```
$ ppr −d thatprn −f "Joseph Andrews" −−charge−to jandrews −X jandrews@somehost.someorg.org
```

This will allow PPR to subsequently distinguish between identically named users on different hosts without choking up the queue listings and banner pages with a level of detail which is generally unnecessary. The `−X` switch is often referred to as the "proxy−for switch" for reasons which I hope are obvious.

In authorization code validation mode, the user name is the one specified in the PostScript `%%For:` line, or if the `%%For:` line is absent, it is the name specified with the `−f` switch. If both are absent, it is the name of the user who invoked **ppr**. In authorization code validation mode, the document must contain a PostScript comment with the correct authorization code. If it does not, the job will be canceled and the user will be informed.

The use name and authorization code may be specified like this:

```
%%For: smith
%TCHCTAuthCode: abr9yg
```

Authorization code validation mode is selected with **ppr**'s `−a` switch.

Banner and Trailer Pages

When many people will be using a printer it is sometimes desirable to identify each print job with an extra page on which appears the name of the person who sent it as well as other information such as the name of the application which generated it and a job title. PPR is capable of printing banner pages as well as trailer pages.

When a user submits a print job he may request or reject a banner page and request or reject a trailer page. However PPR considers the user's request as a suggestion only, making its final descision based on the user's suggestion, and the configuration of the printer.

If bins are defined for the printer, the suitability of the printer's mounted forms for printing banner pages is also a factor. When printing a banner page, PPR will use the currently mounted medium with the highest suitability rating. If all mounted media have a suitability rating of 1, no banner or trailer pages can be printed. If bins are defined for the printer and media is mounted, PPR will adjust the size, shape, and character size of the banner page to suit the medium (paper type) being used.

If no bins are defined, the banner page will be formated for 8.5 by 11 inch paper (or whatever "default medium =" is set to in the [internationalization] section if `/etc/ppr/ppr.conf`), but no attempt will be made to select a specific input tray or to request a specific page size of the printer.

Printer Accounting

Running Accounts

[This section has not been written yet.]

Printer Use Logging

PPR can keep a running log of jobs printed. To enable this logging, create an empty file `/var/spool/ppr/logs/printlog`. One line will be appended to this file for each job which is successfully printed.

Note: prior to version 1.30, the printlog file format was different. Two digits, representing the seconds were added to the time stamp (in the first field). Also, fields 5, 6, 7, and 9 were inserted between existing fields and field 11 was added at the end. The contents of the old fields did not change, only the positions of some of them did. Therefore, existing software will have to be modified to use the new field numbering.

PPR releases since 1.30 have added new fields, so some of the fields at the end may not be present in log files created by old versions of PPR. You should keep this in mind when writing software that parses this log file.

A typical print log entry looks like this:

```
19961115131401,rotate−1375.0(mouse),chipmunk,"David Chappell",chappell,"",7,1,2,166,30.12,0.00,−1
```

The meanings of the fields in the above log line is as follows:

- The 1st field is the date and time at which the job was submitted. In this case the date is November 15th, 1996. The time is 1:14pm.
- The 2nd field is the job id. This job was submitted to the group called `rotate`.
- The 3rd field is the printer it was printed on. This job was printed on the printer named `chipmunk` which is presumably a member of the group `rotate`.
- The 4th field is the name of the person who printed it. This is the name as it appeared in the queue listing and on banner pages. This is a quoted field.
- The 5th field is the name of the Unix user who invoked **ppr** to submit the job.
- The 6th field is the argument to **ppr**'s `−X` switch, if it was used. In this case it wasn't used, so the field contains a string of zero length. This is a quoted field.
- The 7th field is the number of logical pages in the job. This ought to be the number of `%%Page:` comments in the PostScript file. If the number of pages is unknown this field will contain the value −1.
- The 8th field is the number of sheets of paper used. This job was printed 4−Up in duplex mode so only one sheet was used. If 5 copies had been printed, this number would have been 5.
- The 9th field is the number of actual pages (sides) printed. Since this job prints four reduced−size pages to an actual page, this field is 2. If 5 copies had been printed, this field would have contained a 10.

The 10th field is the number of seconds which elapsed between the time the job was queued and the time printing was completed. (To be more precise, the interval between when the program **ppr** was invoked and the time **pprdrv** wrote the log entry.) This job was finished two minutes and 46 seconds after it entered the queue.

- The 11th field is the number of seconds between the time **pprdrv** started up and the time if wrote the log entry. It is expressed in seconds and hundredths of a second.
- The 12th field shows the amount of money charged. This is a fixed point number with two digits after the decimal point. No monetary unit is indicated.
- The 13th field shows the number of sides the printer says it printed. This number will likely be obtained though HP PJL. Note that at this time, the value in this field is not used to compute the charge in the 12th field.
- The 14th field reports the number of pages the printer claimed to have printed over its lifetime. This field is −1 if the number was not retrieved. An attempt to retrieve it will only be made if the printer's configuration file contains a `PageCountQuery:` line which enables this feature and the `feedback` setting is `true`. When this feature is enabled, the page count is fetched just after printing the banner page and just before printing the job itself. If printing is sucessful, the resulting value will ultimately be logged in the 14th field.
- The 15th field is meaningful only if the value in the 14th field is not −1. Once the job has been printed, but before a trailer page is printed, the page count query is repeated. If the query is sucessful, then the difference between the before and after page counts is logged in this field. If the query is not performed or is not sucessful, then a −1 is loggged in this field. See the discussion of the `PageCountQuery:` line in ??? for a discussion of why the value logged in this field may not accurately reflect the number of pages printed.
- The 16th field is the size of the PostScript job in bytes. If the input file was PostScript, then this is the size of the input file. If the job had to be converted to PostScript, then this is the size of the PostScript output of the filter.
- The 17th field is the number of bytes that were sent to the printer in order to print this job. This number may be significantly greater than the number in field 16. For example, it may have been necessary to print multiple copies by sending the job several times or PPR may have inserted fonts into the job.
- The 18th field is the title of the job. If the job had no title, then the name of the input file is used. If there was no input file (such as if the job was read from stdin), then this field will be blank. This is a quoted field.

If left uncontrolled, the printlog file will grow indefinitely, so steps should be taken to truncate it regularly.

User Notification

After a job has been printed, an attempt is made to inform the user. The user is also informed if a job is rejected, arrested, or canceled. The agent which performs the actual communication with the user is called a responder. A responder is a small program, generally a shell script, found in the `/usr/ppr/responders/` directory. No one responder will be appropriate under all circumstances. For instance, different responders are used to send message to Unix shell users and Macintosh clients.

The responder to use is chosen when **ppr** is invoked. It is chosen with the `−r` switch. If no `−r` switch is used, the default responder `write` is used. If you wish to suppress user notification you should use the `−m none` switch.

Among its parameters the responder program is given an address to send the message to. The correct format for the address depends on the responder. For example, for the responder `write` it is simply a Unix user name. For the responder `mail` is it an electronic mail address. For the responder `atalk` it is an AppleTalk network and node number seperated by a colon. The address should be specified with the `−r` switch when **ppr** is invoked. If the `−r` switch is ommitted, the default address, the name of the user who is invoking ppr will be used. This address is suitable for the default responder, `write`. A list of the supplied responders may be found in the ppr(1) man page in the section which describes the `−m` switch. Instructions for writing a responder are found in Appendix ???.

Additional responder options may be specified with **ppr**'s `−−responder−options` switch. These options are a space separated list of name=value pairs. If no `−−responder−options` switch is used then the list will be empty.

You can change the default responder, responder address, and responder options by setting the environment variables PPR_RESPONDER, PPR_RESPONDER_ADDRESS, and PPR_RESPONDER_OPTIONS respectively. For example, if you are using X−Windows, you might want to add the following lines to your `.xsession` file:

```
# Authorize the PPR responder:
/usr/ppr/bin/ppr−xgrant

# Make the xwin responder the default:
PPR_RESPONDER=xwin
PPR_RESPONDER_ADDRESS=$DISPLAY
export PPR_RESPONDER PPR_RESPONDER_ADDRESS
```

Progress Reporting

While a job is being printed, PPR queue listings will show how far along it is. Up to three figures are reported. The first is the percentage of the file which has been transmitted, the second is the number of the page which is currently being transmitted (an ordinal with the first page transmitted being one), the third is the number of pages which the printer reports it has deposited in the output bin. (The number of pages in the output bin will only be reported if the jobbreak method is "pjl" or "signal/pjl".)

The data which **ppop** uses to make these reports is deposited in the job's queue file by **pprdrv**. It takes the form of a line that begins with `Progress:` which **pprdrv** appends to the queue file. This line has three numbers on it. The first is the number of bytes written so far. The second is the number of `%%Pages:` comments written so far. The third number is the number of pages which the printer says it has dropt into the output tray. All three numbers are padded with leading zeros so that as they increase their length does not change.

If the job is not printed sucessfully the first time, provided the `Progress:` line is still the last line in the file, **pprdrv** does not append a new line on the second and subsequent attempts, rather it overwrites the old one. If a queue file contains more than one `Progress:` line, only the last one should be accepted as correct. Multiple `Progress:` lines will generally only be found if a `Reason:` [1] line was appended after a `Progress:` line.

The percentage of the file transmitted, as reported by **ppop**, is obtained by dividing the number of bytes transmitted by the size of the origional PostScript file and multiplying the result by 100. The origional PostScript file is the input file if it was PostScript or the output of the last filter in the filter pipeline. In some cases the percentage indicated when the job is done is somewhat more or less than 100%. Causes such as the alteration of DSC comments, the changing of line termination, and the insertion of feature code will often make a difference of 1% to 5%. The insertion of procedure set such as the one for N−Up printing or the downloading of fonts can result in a total transmission which is several times the size of the origional file. Occasionally a job will be printed which has large unnecessary downloaded fonts. If these are stript out the result will be that only a small amount, say 20% of the job will be transmitted. If switches such as `−s` or `−p` are used to print only a portion of the job the total percentage printed will be reduced accordingly. All of these circumstances are rare, however, so the total printed will almost always be within 5% of the origional file size.

PJL and User Name Display

If your printer and interface support it and you use a jobbreak method of `pjl` or `signal/pjl`, PPR will display the name of the user who submitted the job being printed on the printer's display panel.

If you use PJL with a two−way communications channel to the printer, the printer will report back whenever it drops a page into the output tray. These messages will be used to imcrease the detail of the progress information described in section Section , Progress Reporting .

Media Handling

Modern laser printers very often have multiple input trays. At the same time, version 3.0 of the Document Structuring Convention describes a method whereby the type of medium required for each page in a job may be specified.

When a job enters the queue, PPR attempts to determine what media types are required to print it. If the document contains comments which explicitly indicate this, that information is used. If not, PPR assumes that only one type of medium is required and attempts to deduce what that is. It starts with a default medium. The default medium is defined by "default medium =" in the [internationalization] section of `/etc/ppr/ppr.conf`. The default medium should be US Letter or A4. PPR then looks for comments in the document setup section, such as `%%PageSize:` comments and invokations of the features `*MediaType`, `*MediaColor`, and `*MediaWeight` which would seem to indicate a different sort of medium. Each time one of these comments is encountered PPR revises it idea of what the required medium is like.

Once the characteristics of all the requested media are determined, PPR attempts, for each one, to select a medium known to it that matches the requirements. The list of known types is stored in the file `/etc/ppr/media.db`. For each required medium type, PPR makes multiple passes over the known media list, relaxing its standards each time until if finds a match. If the proofmode is `NotifyMe` and PPR has to relax its standards more than a little, it will reject the job. The list of known media types may be edited with the **ppad media** commands.

For each printer the operator has the option of describing what known media type is available in each bin or of forgoing automatic media handling entirely. The **ppad bins** series of commands tells PPR what input bins are available. If the bin list is empty, PPR assumes that the printer can immediately print any job queued for it, regardless of what media are required. If the bin list is not empty, PPR will not attempt to print a job until all the required media are mounted on one or more bins. In the case of a group, the job will be started on the first member of the group which has the required media in place.

The operator must indicate what type of medium is in each bin by using the **ppop mount** command. The media currently mounted on all the bins of a printer or of a group may be seen by using the **ppop media** command. If a job is not being printed because the required media are not mounted on any candidate printer, its queue status (as indicated by **ppop list**) will be "waiting for media". A list of the required media will also be displayed. Media handling is closely related to but not the same as automatic bin selection which is described in section Section , Automatic Bin Selection .

Automatic Modification of Print Jobs

The PPR spooler has the ability to modify the print job by inserting PostScript code to turn on special printer features such as duplex printing, or to select the bin which contains the required paper. Features or this sort are described in this section.

Inclusion of Feature Code

When **ppr** is invoked, the user may specify the names of printer features which should be invoked. These features are specified with the `-F` switch. The names are those used in the printer's PPD file. These are the same names as appear in the PPD file. Here is an example: to print a file called `mydoc.test` on the printer `myprn` with duplex mode turned on, we would use this command:

```
$ ppr -d myprn -F '*Duplex DuplexNoTumble' mydoc.test
```

The code for the features selected by `-F` switches is inserted in the Document Setup Section of DSC conforming documents. For documents which do not conform, the code is inserted at the begining of the document. The code to insert is taken directly from the printer's PPD file.

When inserted, the code is enclosed in `%%Begin(End)Feature:` comments. When the spooler inserts duplex code, it brackets the duplex code with code to save and restore the current transform matrix since for at least some printers, the duplex code given in the PPD file seems to execute "initmatrix". The two extra lines of bracketing code generated by the spooler end with the comment `%PPR`. When the spooler inserts code requested with the `-F` switch in a DSC conformant document, it inserts it at the top of the document setup section, right after the automatic bin selection code described in the next section, unless it is duplex code. Duplex code is inserted at the very end of the document−setup section. This is done to give it a better chance of overriding duplex setting code already in the document.

Automatic Bin Selection

Automatic bin selection is a feature which is closely related to media handling (see section Section , Media Handling), but it is not the same since it may be disabled for any given job even if media handling is being used. Automatic bin selection will only work if media handling is also used, that is, if the bin list for the printer is not empty.

If a print job requires only one type of medium and automatic bin selection is in effect, then the spooler will insert code in the job to select the paper tray which contains the required medium. (Does automatic bin selection work for jobs with multiple media types? Maybe, but it has not been well tested.)

Since many documents already contains code which influences input tray selection, automatic tray selection is not always successful. This is the strategy used: In the case of non−conforming documents, the code is inserted at the begining of the document. This code will be overridden by any bin select code in the document. In the case of conforming documents, that is, those with a properly commented document setup section, the bin select code is inserted at the beginning of the document setup section. The likelihood of this code's suceeding depends on whether the bin select code the document already contains is properly commented. For example, it works better with Apple's LaserWriter 8 than with earlier versions of the LaserWriter driver. LaserWriter 8 encloses the code which selects paper source and size in standard comments. These comments allow the spooler to modify this code. This is how PPR modifies the code: It removes any `*InputSlot` code and changes `*PageSize` code to equivelent `*PageRegion` code. The spooler removes code by placing a percent sign and a space in front of each line, converting the code into comments. It also changes the DSC comments it replaces into ordinary comments by placing a percent sign and space in front of each line. Thus, you can examine the spooler output and see what code it has replaced. (You can examine the spooler output by setting up a printer with the interface `dummy`. If you wish to know what code is being sent to a real printer, set up the dummy printer with the same options, including PPD file, bins, and mounted media.)

Combined with the automatic bin selection feature is the automatic autoswitch selection feature. If the bin selected is named `Upper` or `Lower`, and there is another bin named `Lower` or `Upper`, the autoswitch selection feature comes into play. If the bin with the opposite name has the same medium mounted, then the automatic tray switch feature is turned on using the code from the PPD file. If the opposite bin has a different type of medium mounted, then the automatic tray switch feature is turned off using the code from the PPD file. (The HP 4 series of printers do not seem to have a command to turn off the automatic tray switch feature. This should be regarded as a defect.)

The automatic bin selection feature may be disabled by deleting all of the defined bins with the **ppad bins delete** command. It may be suppressed for the current job with **ppr**'s `−B false` switch.

Multiple Copies

The spooler can insert code to change the number of copies of a document which are printed. You may ask to have this code inserted by using the `−n` switch when invoking **ppr**.

If the document is DSC conforming, the code to select the number of copies is inserted at the end of the Document Setup Section. If the document sets the number of copies, the code inserted by the spooler, coming later in the document, will override it. Of course, if a job sets the number of copies after the end of the Document Setup Section, this technique will not work. (A document which defines the number of copies outside the Document Setup Section breaks the rules of DSC conformance.)

If a document does not have the DSC comments necessary to identify the Document Setup Section, the code is inserted at the top of the document. This code takes a different form from that inserted in a Document Setup Section: the code redefines **showpage** as a procedure which sets the number of copies before executing the real showpage.

If you invoke ppr with the `−n collate` switch, then PPR will print collated copies. It will do this by either sending the whole document multiple times, for non DSC conforming documents, or by sending only the pages multiple times, for DSC conforming documents. The number of copies desired as well as collated copies may also be requested by means of a `%%Requirements:` comment in the document header if the `−R copies` switch was used when invoking **ppr**.

N−Up Printing

PPR can insert code to cause several virtual pages to be printed on one physical page, possibly reducing the size of the pages in order to do so. This feature may be invoked with the `−N` *`n`* switch where *`n`* indicates the number of virtual pages to be put on each side of the physical page.

Signature Printing

A signature is a booklet−like unit of a book which is sewn or glued together with other signatures to make a book complete. A signature is printed in such a way that the sheets of paper on which it is printed may be stacked and folded in the center. When they are thus folded and glued, stapled, or sewn at the fold, the pages will be in the correct order. PPR can print a DSC conforming document as one or more signatures. 2−Up mode is used to accomplish this. The pages are re−ordered before they are sent to the printer. It is necessary to send the pages to the printer out of sequence so that when the signatures are folded the pages will be in the correct order. Signiture printing is invoked with the `−s` switch. This switch takes an argument which is the number of sheets of paper which should be used to make each signature. This switch also turns on 2−Up mode and duplex mode. The number of pages which will fit in a signature is the number of sheets multiplied by 4.

Signiture printing works best with a duplex printer, however it is possible to print signatures without a duplex−capable printer. The `−s fronts` and `−s backs` switches make this possible. It will be necessary to submit the job once, with the `−s fronts` switch (in addition to the `−s` switch which specifies the number of sheets); take the paper from the output tray, turn it around, and put it back in the input tray, and submit the job again with the `−s backs` switch. This works, but it is tricky since you must first make sure no one else is using the printer, and figuring out which way to insert the paper the second time is not easy.

Booklet Printing

Booklet printing is just like signature printing, except PPR automatically selects the minimum number of signature sheets needed to print the entire document in one booklet. Booklet mode is invoked with the `−s booklet` switch. As with signature mode, booklets can be printed on a simplex printer by passing the paper through twice the first time with the `−s fronts` switch and the second time with the `−s backs` switch.

Page Reversal

PPR can often re−order the pages of a document so that they will stack in the proper order in a printer's face−up output tray. In the case of duplex and N−Up printing, PPR will work back and forth through the document and will generate blank pages where necessary in order to achieve correctly stacked output. The desired output order for a printer may be specified with the **ppad outputorder** command.

Automatic Resource Downloading

Whenever PPR prints a job which contains `%%Include:` comments it attempts to insert the resources requested. There are three sources of resources available to PPR.

One source is the resource library in the directory `/usr/ppr/cache/`. These several procedure sets used by PPR, a few vendor's procedure sets which are included because there are broken versions out there which we don't want corrupting the cache, and a few patched vendor procedure sets.

The second source is the font index. [This section has not been written yet.]

The third source is the resource cache in `/var/spool/ppr/cache/`. Whenever PPR processes a job with an resource it does not have in the cache, it adds a copy of it to the cache. The purpose of this is to reduce resource download time in the future. LaserWriter 8 will generally refrain from downloading a font if the PPR Macintosh printer server **papsrv** informs it that the font is already in the printer. (Of course, it is in the printer, it is in the cache, but from the viewpoint of the Macintosh client it is the same thing.) Also, some PostScript drivers for `Microsoft Windows` allow the downloading of the procedure set to be turned off. As long as one job has been printed with it turned on, the procedure set will be in the cache and if downloading is subsequently turned off, the `MS−Windows` driver will include a comment in each document telling where to re−insert the procedure set from the cache.

TrueType Font Downloading

[This section has not been written yet.]

Font Substitution

Sometimes a font manufacturer changes the name of a font. For example, MonoType's font `MBembo` became `Bembo`. At other times, fonts within the same family will differ only in width. In that case, a reasonable substitute may be achieved by adjusting the width of the substitute font.

If the required fonts are called out with DSC comments, then PPR can automatically substitutes fonts. The file `fontsub.conf` is consulted to help locate a suitable substitute font.

The format of `fontsub.conf` is described in Appendex ???.

The substitution file will only be consulted if the ProofMode is `Substitute`. The ProofMode is Substitute by default. It can be changed by a `%%ProofMode:` line in the document comments or by **ppr**'s `−P` switch.

Printer Interface Programs

The **pprdrv** communicates with the printer through an interface program. An interface program must read the PostScript code from stdin and send it to the printer. If the interface receives messages from the printer it must copy them to stdout.

An interface may also have a rôle in indicating to the printer were one PostScript job begins and another ends. For example, if a flag page is printed before the job it is necessary that the PostScript interpreter be reset after processing the code for the banner page and before processing the code for the job proper. Otherwise, commands executed to print the banner page could have lingering effects which would prevent the job printing correctly. With flag pages printed after the job, the problem is even more severe. If an error occurs in the job, the PostScript interpreter will throw away the rest of the job, and since it has know way of knowing when the job ends and the banner page begins it will throw away the flag page too. Even if the job prints correctly, if it leaves the device space transformed or standard commands redefined, the flag page may not print correctly.

The method used to indicate job boundaries to the printer is known as the job break method. The job break method for a printer may be set with the **ppad jobbreak** command. There are four basic catagories of job break methodes:

There are two unsatisfactory jobbreak methods provided for testing purposes. One is a job break method `none`. As its name implies, it does not attempt to indicate job boundaries. It will seldom if ever prove satisfactory. On a serial connexion, for example, the printer will be unable to distinguish sucessive jobs. With some interfaces it may work because the very act of starting the interface indicates the start of a new job. This may be true of interfaces which invoke Ghostscript or interfaces which make network connexions using certain protocols such as AppleTalk. However even in these cases, the job break method `none` will cause problems if banner or trailer pages are printed or if PPR elects to print multiple copies of the job by sending the whole job multiple times.

The other unsatisfactory method is called `save/restore`. This is only a slight improvement on `none`. When using the `save/restore` job break method **pprdrv** brackets each job with the PostScript commands **save** and **restore**.

The job break methods in the second catagory employ the control character control−d (character code 4) to separate jobs. If the method is `control-d`, **pprdrv** sends a control−D to terminate any old job which may be lingering in the printer and then sends the jobs one at a time, ending each job with a control−D.

When a PostScript printer receives a control−D over a serial interface or, in the case of HP printers with JetDirect cards, over a TCP/IP interface, it finishes processing the job and then responds with a control−D. Thus when all the jobs have been completed, the number of control D's received should equal the number sent. A good printer interface program will count the number of control−D's it copies to the printer and will keep the connexion open until it receives an equal number. Only by so doing can it know that the jobs have been completely processed and that it has received all of the messages (such as PostScript error notifications) that are coming. If the communictions channel is in fact one−way it will never receive any control−D's and will wait forever. For this reason it is necessary to set the queue up corectly with the **ppad feedback** command. See the ppad(8) man page for further information on this topic.

The job break method `pjl` is the same as `control-d` except that it adds HP PJL commands to display the user name on the printer's message panel and to turn on reporting of page completion (see sections Section , PJL and User Name Display and Section , Progress Reporting).

Two more job break methods depend on the interface to indicate job breaks to the printer. These are `signal` and `signal/pjl`. Whenever a job break is required, **pprdrv** finishes sending the current job to the interface and then sends the interace SIGUSR1. When the interface receives SIGUSR1, it reads all it can from **pprdrv**, sends it to the printer and then indicates the job break to the printer. When the printer has acknowledged the job break, the interface will send SIGUSR1 to **pprdrv**. This method is implemented by the interface `atalk` which indicates the division between jobs to the printer by means of a special feature of the AppleTalk Printer Access Protocol.

The final job break method is `newinterface`. When using this method, **pprdrv** runs the interface repeately, once for each actual print job or flag page. This was used by the now obsolete `gs*` interfaces. It may be used in the future to support AppSocket. It will also work with `atalk`, but it is more effecient and reliable to use `signal`

Detailed technical specifications for a printer interface can be found in Appendix ???.

Printer Fault Handling

This topic has already been touched on in section Section , Printer Errors . It is the responsibility of the interface program to connect to the printer, accept the data and send it to the printer. If it cannot accomplish its job it will exit with a code which indicates the reason for the failure. It will generally also append a diagnostic message to the printer's alert log.

One of these codes is EXIT_PRNERR. This code indicates that an error has occured which prevents effective communication with the printer, but the error is of a sort that may clear up. The first time such an error occurs, **pprd** will wait for 10 seconds and then try printing again. After each failure the delay time is increased by 10 seconds until it reaches 10 minutes after which point it does not continue to increase. Errors of this sort may result in alert message being sent to an operator.

Another code is EXIT_PRNERR_NORETRY. This code is similiar to the one above, but it is used for conditions which are unlikely to be self correcting. An invalid printer address or interface option is an example of such an error. These errors too may result in an alert message being sent to an operator.

A third code is EXIT_SIGNAL. An interface should terminate with this code if it catches the TERM signal. When a printer is halted or a printing job is canceled, **pprdrv** sends SIGTERM in order to kill the interface. An interface is not required to have a handler for this signal since the default action is to die, but if the interface has any special shutdown action it must take, it should catch this signal and then exit with the code EXIT_SIGNAL.

A fourth code is EXIT_ENGAGED. An interface should exit with this code if has good reason to believe that the printer is present and turned on but is busy printing a job from another computer or is off−line. When an interface exits with this code, **pprd** delays for 60 seconds and then tries again. No alert notice will be sent to the operator.

Commentators

Note: everything in this chapter will soon be obsolete.

A commentator is a file or external program which is fed information which allows it to keep up a running commentary on the activities of PPR as they relate to a certain printer.

A commentator may be activated full−time for a certain printer. In this case, it will send messages whenever a selected event occurs on a specific printer queue regardless of whose job is being printed.

In contrast, a user can ask for commentary at the time he submits a job. The commentary messages will only be sent while that job is printing and no matter which printer it is printed on.

Activating a Full−Time Commentator

A full−time commentator may be activated by editing a printer's configuration file and adding a line like this one:

```
Commentator: 5 audio yourpc.trincoll.edu:15009 "level=3 voice=male1"
```

or like this:

```
Commentator: 15 samba yourpc ""
```

In the first example, whenever something interesting happens (what is interesting is defined by the number 5) the program `/usr/lib/ppr/commentators/audio` is invoked. In the second example, the number which descibes what the commentator wants to hear about is different. The commentator program is also different: `/usr/ppr/commentators/samba` will be invoked. You may put as many `Commentator:` lines in a printer configuration file as you wish.

In these examples, the first parameter, (5 and 15 in the examples) is a bitmask which defines what we find interesting. You should determine this value by adding together the values which stand for the types of events you are interested in. The value 5 is formed by adding 1, for printer errors, and 4 for printer stalls.

Here are the values which should be added together to form the first parameter:

- COM_PRINTER_ERROR 1 Printer status messages when are prefixed with "PrinterError: ".
- COM_PRINTER_STATUS 2 Other printer status messages.
- COM_STALL 4 Printer is accepting data too slowly.
- COM_EXIT 8 The printer driver program (**pprdrv**) has exited due to some unuasual circumstance.

Whenever an even occurs and all of its categories to which it belongs to are specified by the number in the first field of the `Commentator:` line, the commentator program specified in the second parameter is invoked. If no path is specified, the commentator program is assumed to reside in `/usr/ppr/commentators`.

Activating a Commentator for a Specific Job

[This section has not been written yet.]

How Commentator Programs Work

The commentator program is invoked with the following arguments:

1. 1) The address from the third field of the `Commentator:` line.
2. 2) The commentator options (`Commentator:` line field three).
3. 3) The name of the printer.
4. 4) The category value for this event.
5.

5) A very brief text message describing the event.

6.
   6 and 7) An unprocessed or raw form of the information in the text message. At times the raw data fields will contain information which is not preserved in the fifth argument.

7.
   8) PPR's assessment of the serverity of the situation. This assessment takes the form of a number between 1 and 10 with 1 being not at all serious and 10 being very serious (something that will probably require the attention of the system administrator.)

8.
   9) A longer text message describing the problem in complete sentences. This messages can be used by simple commentators instead of creating their own messages from the information provided by means of the arguments described above.

Thus, a commentator might be invoked like this:

```
commentators/audio yourpc.trincoll.edu:15009 'level=3 voice=male1' yourprn 4 'stalled for 12 minu
```

when the printer has not been accepting data or like this:

```
commentators/audio yourpc.trincoll.edu:15009 'level=3 voice=male1' yourprn 4 'no longer stalled'
```

when it begins accepting data again, or like this:

```
commentators/samba yourpc "" yourprn 8 'has printed a job' EXIT_PRINTED "" 1
```

whenever a job is printed, or like this:

```
commentators/samba yourpc "" yourprn 8 "job error" EXIT_JOBERR ""
```

if there is something wrong with a job, or like this:

```
commentators/samba yourpc "" yourprn 8 "PostScript error" EXIT_JOBERR ""
```

when the specific thing wrong with the job is a PostScript error, or like this:

```
commentators/samba yourpc "" yourprn 8 "interface program killed" EXIT_SIGNAL ""
```

when somebody kills the interface program, or like this:

```
commentators/samba yourpc "" yourprn 8 "printing halted" EXIT_SIGNAL ""
```

when PPR killed the interface program itself, or like this:

```
commentators/samba yourpc "" yourprn 1 "out of paper" "Out of Paper" 0
```

In this example, the first raw data field contains a message received from the printer. The normal message field contains a standard version of this message. (Believe it or not, printers don't all phrase these messages the same.) The various forms are converted to standardized forms using the information in the file `/etc/ppr/lw_messages.conf`.

```
commentators/samba yourpc "" yourprn 1 "" "Fatal error 1001" 0
```

Notice that in this last example the field which ought to contain the message for the user is blank. However, first raw data field contains the message exactly as it was received from the printer. Apperenly, PPR did find a standard form of this message in its data files.

Also notice that in the last two examples the second raw data field is blank. The second raw data field is currently blank for this type of message but in a future version of PPR will contain the translated version of this message from the printer's PPD file. The language will be indicated by the first word of the message.

If the program name in the printer configuration file is `file`, the event is appended to the file which is named in the address field. When the file is opened, the PPR home directory, (usually /usr/ppr) is the current directory. You can rely on this or you can begin your file name with a `/`. If the commentator program name is not `file` and does not begin with a `/`, it is understood to be a program in the `/usr/ppr/commentators/` directory. If you want to use a program in another directory, you must begin its name with a `/`. Commentators should exit as soon as possible because **pprdrv** will wait for up to 60 seconds for all the commentators it has launced to exit. If necessary, commentators should place their work in the background or in a queue and then exit.

The Samba Commentator

This commentator uses Samba's **smbclient** to send a popup message. The address is the NetBIOS name of the machine to send the message to. If you think **smbclient** is going to have trouble determining the IP address, you can append a hyphen and the IP address.

The X−Windows Commentator

This commentator uses **xmessage** or **xterm** to display a message. The address is an X−Windows display name. The user `ppr` must have permission to create windows on the display. This can be granted with **xhost** [2] or **ppr−xgrant** [3] .

The Audio Commentator

This is the most elaborate commentator by far. It makes spoken announcements concerning the printer's condition. To use the audio commentator you must have Perl 5. You must also install the sound files. To install the sound files, get `ppr−speach−1.30.tar.gz` and unpack it in the PPR home directory (which is usually `/usr/ppr/lib/`). To set up the audio commentator you need a commentator line like this one:

```
Commentator: 13 audio /dev/xxx level=8
```

Or like this one:

```
Commentator: 13 audio smith.pc.myorg.org:15009 level=10
```

Both these lines This will cause events of certain types (indicated by the bitmask 13 or 31) to be reported using the audio commentator. The commentator is also passed the option `level=10` which it uses to furthur filter the events which have been selected by the bitmask. A level of 10 will result in the reporting of all events, even the sucessful printing of a job. Lower numbers will exclude mundane events. Remember, the bitmask is formed by summing the desired catagories of messages. The catagories are:

1
: printer errors, such as paper jams

2
: printer status messages such as off line

4
: notices that the printer isn't accepting data

8
: **pprdrv** exited, returning a code which indicates an abnormal condition such as communications error or PostScript error or the job was canceled

16
: **pprdrv** exited but for a rather booring reason, such as "job was printed."

The address indicates the means of playing the sound file. The current version of the audio commentator provides two ways to do this. If the address begins with `/`, as in the first example, then the sound is played on the local machine. The code to accomplish this is in the file `/usr/ppr/lib/play_local.pl`. This code is presently very redimentry and requires a program which will take the name of the file to be played as its argument. As distributed, the program `/usr/local/bin/play` is used and the address is ignored. Because the address is ignored, the silly address `/dev/xxx` is used in the example above. Any help with improving this would be greatly appreciated.

Whatever method of playing the sound you select, the easiest way to test the commentator is to set the bitmask to 31, the options to `level=10` and print a job. You should hear something such as, "The printer emm cee ee cee underscore one has just printed a document."

Input Filters

One of the most noteworthy features of PPR is its ability to determine the type of the input file and automatically convert it to PostScript if necessary. The type of the input file is determined by two methods. The first method is the most reliable. The second is employed only if the first fails.

The first method is to look for a "magic number" at the start of the file. Many file format specifications dictate that files conforming to them always begin with certain signature bytes. Formats which specify a signature or magic number include DSC conforming PostScript, JFIF, GIF, PNG, DVI, WordPerfect documents, PBM, XPM and others.

If no recognized magic number is found, then the first 8192 bytes of the file are scanned and the number of times certain characters and constructs are found are counted. The things counted include ASCII control characters, non−ASCII characters, HP escape sequences, Troff−style dot commands, TeX style backslash commands, and PostScript procedure definitions. When these things have been counted, the totals are used to try to guess at the file format.

If PPR ever fails to determine a file's format correctly or you wish to print a file in a format for which auto−detection is not available, you can override the auto−detection mechanism with **ppr**'s `−T` switch [4] .

Once the file format has been determined, unless it is PostScript, it is necessary to pass the file through a filter. A PPR filter is a program which reads the file from its standard input and writes PostScript on its standard output. It is the PostScript code emmited by the filter that PPR sends to the printer.

A number of filters are supplied with PPR. PPR also includes a number of shell and Perl scripts which work together with programs such as TeX, Troff, PBMPlus, and other programs often found on Unix systems to form filters. If the supporting programs can not be found, the script filters will not be installed.

PPR can automatically recognize a number of file types for which it does not have filters. PPR determines whether it has a filter for a file of a certain type by looking in the `/usr/lib/ppr/filters/` directory for a file with a name in the form `filter_`*`type`*. For instance, if the input file is of type `fortran`, it will look for `/usr/ppr/filters/filter_fortran`. You may write your own filters and install them in the `/usr/lib/ppr/lib/` directory. For explicit information on writing filters, see Appendix ???.

Each filter receives a list of filter options. This option list is created by concatenating the default filter options with all of the filter options specified with **ppr**'s `-o` switch. Before invoking the filter, **ppr** culls the option list. It removes any option which is intended to apply only to a specific filter other than the one being used. An option may be applied to a single filter by prefixing its name with the name of the filter and a hyphen. For example, the option `noisy=yes` should apply to all filters and hence won't be culled but the option `dvi-noisy=yes` will be deleted by **ppr** except when it is invoking the DVI filter. A filter, such as the `tex` filter, may invoke another filter such as the `dvi` filter to do part of its work. The first filter will generally pass its options on to the second filter, however, since the filter list has already been culled, the DVI filter will receive options whose names begin with "tex-" but will not receive options whose names begin with "dvi-" [5] .

What follows is a description of some of the filters supplied with PPR. [6]

The Line Printer Emulator

[Not written yet.]

The pr Filter

The **pr** filter passes the input through the Unix **pr** program and then through the Line Printer Emulator. It supports the options `width=` and `length=` which are passed to **pr** as the arguments to its `-w` and `-l` switches respectively. Since the this filter invokes the Line Printer Emulator, it supports all the options of that filter too.

This filter will never be invoked automatically since plain text files are normally processed by the Line Printer Emulator. In order to employ this filter you must invoke **ppr** with the switch `-T pr`.

The Dot Matrix Printer Emulator

[Not written yet.]

The Fortran Filter

This filter prints files encoded with fortran carriage control. This filter supports the options `width=` and `length=` which specify the page width in columns and the page length in lines respectively.

Various Image File Filters

Depending on which image file utilities your system has, filters may be available for various image file formats. These filters are shell scripts which invoke programs such as the PBM utilities and the Independent JPEG Group's programs. These filters will be installed and edited to fit your configuration if the required utilies are in the PATH when you invoke **ppr_indexfilters**. These filters support some options, but they are generally ones such as `resolution=` and `color=` which PPR supplies automatically as default filter options.

The Hex Dump Filter

This filter prints side−by−side hexadecimal and ASCII dump of the first few hundred bytes of the input file (as much as will fit on one page). This filter will sometimes be used to print input files for which no filter is available. This filter will only be used if either the `−e hexdump` switch was used when invoking **ppr** or if it was not possible to inform the user of the problem by means of a message on stderr or by invoking a resonder [7] .

The TeX Source Filter

This filter is written in Perl, so it will not be available if Perl was not installed last time you ran **ppr−index filters**. This filter will read the input file and attempt to identify it as either Plain TeX or LaTeX. It will then copy the file to a temporary directory and run TeX on it with either the Plain or LaTeX macros. It will run TeX multiple times if it is necessary to get the cross references right.

If running TeX results in a fatal error, TeX's output is passed through the Line Printer Emulator and becomes the output of the filter and hence what gets printed. On the other hand, if TeX ran successfully, the DVI file is passed to the DVI filter.

This filter has one option, `noisy=`. If `noisy=` is set to `true` then progress messages and the terminal output of TeX are printed on stderr. The `noisy=` option is also passed through to the DVI filter. (This filter passes all of its options on the the DVI filter, the DVI filter will ignore those it does not understand.) Remember, if you intend an option to apply only to this filter and the instance of the DVI filter which it invokes, you should prefix "tex−" to the option name even if you want the option to be passed through to the DVI filter. For example, if you invoke **ppr** with the option `−o 'noisy=false tex−noisy=true'` then the DVI filter will be quiet if you submit a DVI file but will print noisy messages if you submit a LaTeX file to **ppr**.

The Texinfo Filter

This filter is a shell script which converts documents from the Texinfo format used by the GNU project to PostScript. It first passes them through texi2dvi (which you must provide). It then passes the resulting DVI file through the DVI filter.

This filter supports the option `noisy=`. If one uses a command such as:

```
$ ppr −d aardvark −o noisy=yes
```

a running commentary will be printed as the file is converted. This commentary includes whatever TeX sends to stdout. The `noisy=` option is also passed on to the DVI filter, so it produces lots of messages too, including the terminal output of DVIPS.