# Perspective On Printing

Ben Woodard
*VA Linux Systems*
ben@valinux.com
Nick Moffitt
*VA Linux Systems*
nick@valinux.com

## Abstract

Printing is one aspect where Linux has progressed very slowly. Most free operating systems continue to use Berkeley LPR, and replacement printing systems have been slow to catch on. This paper describes how VA Linux Systems and HP's LaserJet division teamed up to add PPD functionality to lpr.

## 1 Introduction

In the world of computing there are certain topics which are more sexy than others. For example, kernel development is sexy because the people that do it regularly are very elite. User interface development is sexy because you are doing things that people see and you have the opportunity to make something flashy and visually appealing. In between the kernel and the user interface are a whole bunch of command line tools and libraries that have very little sex appeal within the hacker community. Even within this world of command lines and APIs some things are more attractive projects to work on than others.

The system daemons are amongst the most often ignored, the most often neglected pieces of software on a system. Many of these daemons were written in the early days of BSD 4.2 by grad students and never have really had an owner or maintainer. They were nothing more than check boxes on the list of milestones that the CSRG had to complete to satisfy the requirements for its grant money. They were often banged into existence over a Christmas vacation to meet a DARPA deadline by grad students who had varying degrees of skill, and then cobbled together into something that sort of worked. Since that point the only time that people look at them is when is when someone discovers a security problem.

This is the story of LPR.

### 1.1 History

In the very early days of Unix, there were no differences between terminals and printers, there were only teletypes. Your shell prompt was on the teletype and the output of your commands was always printed. It didn't matter if the output was a directory listing or the results of some long running job. If you needed a print out to give someone, you simply got our your scissors and cut the paper in the appropriate places and gave that to whomever needed it. The CRT terminal changed that, and the path taken by terminals and printers diverged forever. Now we have fancy translucent x-windows based terminals and high resolution network connected laser printers. They seem about as different as you can get but they both have a common ancestor.

One of the first changes was that printing became batch oriented while terminals remained interactive. This was the birth of LPR. Initially, there must have been a bunch of development of LPR as everybody added the features that they needed. The impression that I get having talked to Kirk McKusick was that someone wrote LPR in an afternoon and then pretty much abandoned it. Since the source code was around, various people added the features that they were interested in having, but there was never a central person who maintained the whole thing. At some point, LPR did enough things that people were happy with it and turned their attention elsewhere. At that point LPR basically underwent a feature freeze from which development never resumed.

About that time, Unix fragmented and everyone picked up their copy of the BSD source code and ran with it. Every one made fixed the bugs that they found and added the features that found necessary and various incompatibilities arose. This made printing very difficult between different versions of Unix and so everyone kind of standardized on the least common denominator version of LPR and refused to alter it for fear of breaking compatibility. This helped to freeze LPR even more solidly.

The problem is that since that time, printers have evolved from simple teletypewriters with potentially 3 or 4 different font wheels directly connected to a computer through a serial port into high resolution, high quality, network connected printers. However, LPR has failed to evolve with them. The memory of the problems caused by the fragmentation of Unix and the resultant fragmentation of LPR still lingers on and very few people are willing to delve deeply into LPR to bring it out of its frozen state. During this time rest of the world has not stood still. There have been at least two major paradigm shifts during that time, personal computers and the internet. LPR and the rest of the Unix printing infrastructure has also failed to make the transition to meet the demands of the current printing audience.

Because of its lack of sexiness, because of fears grounded in the past, because techies don't print as much as traditional users, and because of a lot of other reasons Linux printing is very primitive and needs to be brought up to date with current technologies.

All these years of neglect have allowed many issues to build up until now, and there are problems all over the place. Several people have worked on different parts of the solution but it has yet to come all together.

## 2 VA begins work

In the past I worked for Cisco Systems and I helped construct the print environment there. We were able to take standard components such as LPR, Samba, and Apache and create a very scalable, highly reliable, print system. During that time, I built up a lot of experience working with printers and even more ideas of how to make printing work better. I changed jobs and moved over to VA Linux Systems and a few months later HP's LaserJet division, came knocking on our door. They wanted someone to work on improving printing for them.

They had done their homework. They knew that printing wasn't a sexy project that people were excited to work on. If I remember correctly they referenced Eric Raymond and said, "You have to pay people to take out the garbage." They had an idea of what facilities were most urgently needed and were willing to open up their checkbooks to make it happen.

## 3 Libppd

One of the most pressing problems with respect to Unix printing for printer vendors these days is that they build all these wonderful capabilities into their printers but no one in the Unix world has access to them. This problem bites just about everybody. The users have a problem because they really can't make use of these features. The sysadmins have a problem because to fill the users' needs they have to do a lot of work in the filter scripts to enable the necessary features. The application software developers have to write functionality that allows them to access the printer's capabilities. The printer vendors have two problems. They can't use these features to distinguish their products because the features are of little value to a user community that can't make use of them. The second problem is since the application developers have written their own support for printers, and some of them have done it improperly, often causing the users to blame the printer vendors.

Our solution to this problem was to make a library called libppd. Because the LaserJet division was paying our bills and because the low end printers are a moving target, Libppd is designed for PostScript-capable printers. The idea is that libppd and the associated utilities would provide solutions a solution for almost all the people affected. User's would have access to the features that they wanted. Sysadmins would find it easier to write the filter scripts that they need to satisfy the users' requirements. The application developers would be saved the effort of writing their own code to generate a PPD parsing engines. And HP and other printer vendors would have applications that more reliably show off the fancy features that distinguish their printers from other vendor's printers.

### 3.1 PostScript and PPD files

PostScript is designed to be a completely device independent language. It describes the pages in terms of high level abstract picture elements such as fonts and letters, lines and curves rather than pixels. It expects the printer to map these drawing elements onto whatever underlying imaging system it might have. This way an application can generate PostScript and not have to worry what kind of printer it is going to be sent to. You can send exactly the same PostScript job to an old 300 dpi ink jet printer as well as a a 2400 dpi image setter. The only difference is the quality of the output.

There are many advantages to this model of printing that are not commonly recognized. First of all, the fact that PostScript files can be sent to any printer means that the print job can be archived and printed at a later date without having access to either the application that created the document or the printer that it was first printed on. The more paranoid amongst us believe that binary device specific printer languages are yet another attempt by the legacy proprietary application vendors to separate users from their data. This means that you need to always have an available copy of Word and an operating system that can run Word in order to print a Word document. Other device specific printer languages are not archivable because they require that a specific type of printer always be available.

Since PostScript files are device independent, PostScript provides a mechanism for vendors to specify device specific commands to be inserted into a device independent PS file just before printing. PostScript Printer Definition (PPD) files store device specific commands as well as some hints about the user interface that should be used to present them to the user. In the model that Adobe envisioned when they designed PostScript, an application would output device independent PS which could potentially be archived. Then just before the print job is sent to the printer, the spooling system (which knows what kind of printer it is sending the job to) uses the appropriate PPD file to look up the device specific PostScript commands necessary to enable a particular feature and then inserts them into the PS job at appropriate locations.

### 3.2 Development of libppd

It has been said that "good programmers write good code. Great programmers steal code." I'm not saying that I'm a great programmer but will confess to being very lazy. Parser code, which would be needed to parse a PPD file and manipulate PostScript jobs based upon the contents, has got to be one of my least favorite types of code to write. So I looked around to see if anyone had written a PPD parser already. I was lucky enough to find that the developers of CUPS had written just the kind of thing that I needed. So I ripped out their PPD processing code, tweaked it a bit so that it would stand on its own, and *voilá*, I had libppd. libppd hasn't evolved very far beyond it's CUPS origins, and still has some vestigial remnants that need to be cleaned up. Despite this, it performs its intended function nicely.

As a sort of combination test and prototype application, I brought over another program from CUPS. Under CUPS it was a deeply buried filter called pstops. I modified this to take more conventional arguments and also to be used not only as a system level filter but also as a filter called by users. I changed the command-line arguments and gave it a new name, "ppdfilt", so that it wouldn't conflict with another program that I had. The purpose of ppdfilt is to take a PostScript file and a list of PPD options and modify the PostScript job to incorporate the device specific PostScript commands from the PPD file.

### 3.3 Modifications to LPR

The initial cut of ppdfilt allowed me to take a PS job and run it through ppdfilt and have a device specific PS job with the appropriate options come out the other side. As a user tool ppdfilt is useful; however, because it is not integrated into the spooling system, it requires extra work on the part of the users to print. They must first manually convert whatever they have into PostScript rather than allowing the filter scripts to do the conversion automatically. Secondly, they must pipe their PS through ppdfilt before handing it off to LPR. This is not very difficult but it does involve more typing. Finally, they must also know what kind of printer they are printing to and where the PPD file for that printer is stored on their systems. In addition, this processing of the PS file before it is sent to the spooler doesn't jive with the way that Adobe envisioned printing with PPD files to work.

The problem of integrating ppdfilt into the spooling system is a slightly difficult one. Somehow we have to pass information about the user's desire for certain features from the lpr command line to the print filter. LPR, whose options have not been significantly changed since the days of the four font wheel teletypewriters, does not have any command line arguments flexible enough to allow PPD options to be passed to the print filter. To fix this, a new command line argument was added, '-o'. Once again I stole some code, this time from my old friend Damian Ivereigh who made similar modifications to the the version of lpr that they run at Cisco.

We cannot modify the print job based upon the '-o' option because that could potentially break the file format conversion process, printers that do not use PPD's filters that know nothing about the '-o' option or printing of files by symlink reference. Therefore we must make changes not to the print job's data file, but rather to the print job's control file. Since the format of the control file has not changed since the the last time lpr's command line options changed, there were no existing tags which were suitable for passing this new information. Fortunately, the lpd daemon ignores any items in the control file that it doesn't know what to do with. We simply added a new type of item 'E', for environmental variable, and backward-compatability was maintained.

When you specify one or more '-o' options on the lpr command line, lpd adds one or more E items to the control file. When the LPD daemon comes across these items, it appends them to an environmental variable, LPOPTS which is set in the context of the print filters. That way only filter scripts that want to know about the options passed on the command line have to be modified to handle the extra information.

There are some security considerations to adding this new '-o' option. You are taking information from an untrusted source, the lpr command line, and making it available while running with some privilege, the context of a print filter. This requires some care. To make it difficult for a malicious user to exploit this, the characters allowed in the arguments to '-o' are limited to [0-9A-Za-z:=]. A maximum of 50 '-o' options are allowed. I would be very interested in having some people with more experience writing secure software evaluate this patch to make sure that there are not any issues that I missed.

### 3.4 Modifications to rhs-printfilters

Once I had the means of passing information from the lpr command line to the filter script, it was an easy step to incorporate it this into the rhs-printfilters on my system. The only difficulties were making sure that the PostScript processing happened after the file conversion had completed and making sure that it didn't interfere with non-PostScript printers or printers for which we didn't have a PPD file.

The design of rhs-printfilters made this fairly easy. After all the file conversions are done, if it is a PostScript printer, it executes the ps-to-printer.fpi script. This eliminates the problem with all non-PS printers. To ensure that it didn't adversely affect any printers for which there wasn't a PPD file, I don't feed the print job through ppdfilt unless ppdfilt exists, LPOPTS is set, and PPDFILE is set. LPOPTS is set by the LPR modification mentioned above. PPDFILE is added to the configuration file postscript.cfg, which (when using rhs-printfilters) resides in the spool directory. To make sure that this line ends up there automatically required changing the way that printers are normally setup when running the Red Hat printtool.

### 3.5 gpr

In traditional open source fashion, a few weeks after I had released the initial code, someone contributed to it. A guy named Thomas Hubbell of CompuMetric corporation, a company that provides vendors with usability information about printers, felt that having to put all those -o options on a lpr command line was too cumbersome and so he sat down and wrote a gnome based GUI to my '-o' enabled version of lpr.

It was a nice contribution. It greatly simplified the process of picking what options you want for a print job. I, a staunch command line kind of person, found it so much easier to use that I started running it regularly to test various features within the printer.

Right now gpr is a stand alone program that you run when you want to print. It pops up its own window; and once you are done selecting the options that you want, it executes lpr with the correct set of options. I cannot speak authoritatively for Thomas (and it is his program) but when we first conceived it, we thought that gpr would make a good print-setup dialog box that could be incorporated into any gnome program that needs to be able to print. The thing was that neither Thomas nor I knew how to make a component like that when he first suggested the project.

## 4 Setup, configuration, and printer management

All the work in creating libppd, and ppdfilt, and modifying lpr and rhs-printfilters made it possible to easily send print jobs to PostScript printers with special device specific commands. However, that was only half of what HP wanted us to do in our initial attempt to make printing work better for Linux. HP also wanted to make setting up printers and configuring them substantially easier. This is an amazingly broad topic because Linux lags so far behind other operating systems in this area. Obviously, we couldn't attack all of the many facets of this problem at once. We picked a couple of pieces of low hanging fruit.

Back in the days of the teletypewriters, pretty much everybody was using the same computer through physically connected terminals. A Unix machine would have literally hundreds of serial ports and the teletypewriters would be connected to some of them. Because these units were physically connected to serial ports, and because that physical connection was the job of professional system admins who had a deep understanding of both the hardware and software, it was not unreasonable to also ask the admins to configure the software to use particular printers. They installed the printer, connected it, and so would know how to configure it. Things have changed a bit since then. The average Unix box is much smaller and instead of having multiple sysadmins taking care of one box you usually have one sysadmin taking care of multiple boxes. The people connecting the printers are not usually professional sysadmins; they are average users who have to admin their box because there is no one else to do it. Printer vendors are usually targeting different audiences these days. Printers are either sold as consumer devices with little or no technical documentation, or they are sold to business users where a little more documentation is available but is written for IT professionals. Many printers these days are not connected directly to the Unix machines, exist as standalone network devices.

### 4.1 Printtool

Printing under Linux has always been a bit of a cumbersome thing to setup. The documentation on how to set it up is pretty scarce and to be honest, it is an involved multistage process. Since the software is basically there but the difficulty is in the configuration, it has generally fallen on the shoulders of the distribution maintainers to make the setup and configuration of printers easier.

A few years back, Red Hat took a stab at improving the setup and configuration of printing when they made printtool. Printtool is a TCL/Tk application that does most of the grunt work of setting up a printer. At the time that printtool was originally written, the standards that allow printers to be automatically detected and configured were still nascent. Since that time, the standards have solidified and have become more widely deployed, but printtool hasn't been updated to use them.

Red Hat hasn't been completely asleep at the switch, though. They have been tinkering with the possibility of replacing printtool with a new KDE-based application with some capability to automatically detect printers. This application has yet to see the light of day and may never be fully functional.

We decided that a good first step would be to simply modify printtool to enable automatic detection and configuration of printers. TCL/Tk being what it is, this was fairly simple. We modified the user interface to provide a means to automatically detect the printer, and triggered a command line utility called "pconfdetect" which actually did the work of detecting a printer.

### 4.2 pconfdetect

Having a separate program pconfdetect do the work of detecting a printer seemed like a wise idea to us for several reasons. Being true Unix people, we believe that GUI's are nice but they are not easily scriptable and that makes them difficult to embed in other applications, and often limits their scalability. Seeing that Red Hat had already taken a stab at making a replacement for printtool told us that printtool might not be long for this world and a having lots of work invested into printtool might not be efficient for over the long run.

Finally, TCL/Tk is a good rapid application development tool, but is not the best language for doing the kind of low level programming that is necessary for printer detection. Having TCL/Tk call C functions is possible, but is more difficult than executing an external program and parsing the results. Also someone someday might decide to develop a new printer configuration utility and it would be much easier for them to embed pconf detect than to steal a bunch of code out of printtool. Finally, pconfdetect is really a very thin wrapper around some functionality we bundled together into something that we called libprinterconf. It was easier to develop and test libprinterconf with a simple command line tool like pconfdetect than to try to develop and test it already embedded into printtool.

### 4.3 libprinterconf

Right now libprinterconf is just the library of functions that actually do the work of detecting the printers. It was conceived with a much broader scope in mind. The original notion was that it would be a library of functions that people writing printer configuration utilities could call to do most of the work for them.

The first feature we implemented was two different kinds of printer automatic printer detection. We are planning to add two more significant features to libprinterconf. We intend for it to setup spool directories, and printcap files and anything else needed by a particular spooling system. We also intend for it to be a clearing house of functions for configuring not only the spooler aspect of a printer, but also for configuring the printing hardware.

Auto-detection of printers can be done quite a few different ways. For printers connected to the parallel port there is a protocol called IEEE 1284 which can be used over a bidirectional parallel port to query any attached hardware for certain types of information such as what it is. For network connected printers, you have several options to auto-detect printers. For IP you can use SNMP to detect printers. HP also implements a multicast protocol something like SLP but markedly different. You can also detect printers using the IPX and Appletalk protocols. Printers can also be connected serially but there is no standard way to identify them. Some printers have IR ports and there is a protocol within the IRDA standard that facilitates discovering and identifying IR devices. Finally, USB has its own method of identifying hardware. For this first swipe at automatic printer detection, we implemented the IEEE 1284 parallel port detection as well as SNMP detection. We tried to implement the HP SLP-like protocol but we were unable to reverse engineer it.

Like most things in this project, we stole code from other places. The parallel port printer detection code came from the KDE printer setup tool that Red Hat made but never really put into production. The SNMP code came from a package that I had previously written called npadmin.

### 4.4 snmpkit

npadmin was a program I wrote a few years ago while I was at Cisco. It used SNMP to find out all sorts of information from SNMP capable printers. When I was writing it, I discovered that the SNMP library that existed at the time could not do the things that I needed to do. The problem was one of performance: I needed to poll 3000 printers and find out information from each of them. Quite a few printers were either turned off, had been retired, or for some other reason were unavailable on the network. If I were polling these printers sequentially, the time added to the run by having to wait for the timeouts on printers that were down was unacceptable. I needed something that could plow on through a large number of printers without waiting for down printers. I also found that just forking off new processes for each printer used an unacceptable number of file descriptors. When I designed npadmin, I made it multiplex the SNMP sessions on one socket. The initial design was a very clever state machine but I soon figured out that SNMP interactions were arbitrarily complex and making a state engine flexible enough to handle this arbitrary complexity was extraordinarily complex. I finally decided to shove most of this complexity off onto the kernel's scheduler and implemented npadmin using threads.

It had always been my intent to make a separate library out of the SNMP portion of npadmin, but I had never taken the time to do it. Uncoupling snmpkit from npadmin turned out to be a much bigger project than I ever imagined. The work paid off, though. I believe that I made a nice clean SNMP library optimized to do large numbers of transactions quickly without putting undue burdens on the operating system. Snmpkit was originally written in C++ but to make it more useful to the UNIX community, I also added a C library interface to it.

# 5 Current gaps

This printing project for HP was a first pass at improving printing under Linux. It was never intended to be the complete solution to the many problems with Unix/Linux printing. It was just intended to get the ball rolling and pick off some low hanging fruit. There are many gaps and weaknesses in the work that we have done. Whether these gaps ever get filled depends on a lot of things, time, money, community interest and passion.

## 5.1 Libppd and ppdfilt

Libppd was ripped out of CUPS, and I will confess that the edges are still ragged. It needs to be cleaned up so that it looks less like something ripped out of another piece of software and more like a library of its own.

The containers in libppd are something that I have always found problematic. Heterogeneous data structures are stored in unterminated vectors with associated counts. Although this works, I think that the code could be greatly simplified and potentially made more robust if it we used glib to store the groups of objects.

My original intent was to rip libppd out of CUPS and then put it back as a shared library that both CUPS and other utilities relied on. For various reasons this never happened, but is a possibility for the future.

Many printers are not PostScript printers and therefore do not have PPD files. There is a standard called UDMF evolving that seeks to do something like PPD files for all types of printers. When this standard has settled down and is widely deployed it might be worth it it to turn libppd into libudmf.

## 5.2 GPR

I have always been pleased by the existence of GPR. To me it has always seemed like a bonus. Thomas Hubbel wrote it without any help from me, so it has never been *my* program and probably never will be. Therefore, I don't directly control its destiny. This does not stop my imagination from extrapolating where I would like to see it go.

I would like to see it evolve into the standard gnome print-setup dialog box. That way every application that needs to print will have access to these features.

Right now many of the PPD options are all lumped together on the "advanced" page of the gpr interface. Despite the fact that the PPD specification suggests that the authors of PPD files define groups for their options, very few ever have. Thomas and I have kicked around the idea of making gpr implement the groups feature and then adding groups to all the PPD files that we use and making the modifications to other PPD files an open source project that people could participate on. This would clean up the gpr interface quite a bit and make it more useful.

There is no doubt that GPR simplifies the process of selecting the PPD options when printing. All of these options are available through the command line, but they are a bit cumbersome to use. There are a couple of ideas we have about using gpr to simplify things for command line users. The first is to make it possible for the users of GPR to view and copy the command line that gpr will execute upon completion. That way users will be able to take that command line and modify it or embed it in scripts. The second idea involves the creation of a ".gpr-aliases" file. Whenever you create a saved set of options within gpr, it will write an alias in ".gpr-aliases" so that if you source the file out of your .bashrc or similar file then you will have quick access to your saved settings from the command line.

## 5.3 LPR

As I have said before, LPR is a very old program that is desperately in need of being seriously updated. In this project, we intentionally modified LPR as little as possible under the assumption that it would be more likely that the changes would be adopted. Whether this was a valid assumption or not remains to be seen. There are a lot of improvements that need to made to lpr.

Quite a few of the things we are planning to do are just cleanups which should have been done a long time ago. We need to make the software autoconfed and automaked and deprecate of obsolete options.

Other, planned changes are to provide better interoperability with other programs. Instead of having programs like GPR and Samba being forced to parse the printcap file for themselves, we should make a library out of the parsing routines that LPR uses and then export that interface. In addition to functions to read the printcap, to simplify the writing of configuration applications, we should provide functions that *write* the configuration files.

Once those are done, we rewrite LPR to use this library and then provide alternate versions of the library that work completely differently. For example, we could pull out the library which reads from printcap and replace it with a library which reads the configuration from an LDAP directory or something. Maybe we'll call this libprintspool or something like that. If we make the API general enough, then it would give people the option of selecting spoolers as well. A sysadmin could plug in LPRng, CUPS, or what-have-you so long as the spooler interacts with the underlying data store using the libprintspool functions.

Still other changes that we are considering would be a radical departure for LPR. Why not make LPR use SLP to announce its presence? Why not implement IPP as well as the standard LPR protocol?

### 5.4 npadmin

There are only a handful of things that I am going to do with npadmin in the future. I am going to port it over to using snmpkit, as the npadmin SNMP code is a little old. I am going to finish implementing all the SNMP sets that are in the RFC 1759 MIB. I should have done this a long time ago but I just never got around to it. The third thing is I am going to restructure npadmin in such a way that most of the functionality is in libprinterconf and that npadmin in and of itself is just a thin wrapper on the libprinterconf functionality.

### 5.5 snmpkit

Snmpkit has a few gaps in it as well. It needs to have set functionality added to it and will probably have to be updated to incorporate the SNMP3 specification if printer vendors want it.

### 5.6 printtool

Maybe we will rewrite printtool to make it a nice gnome application and have it use libprinterconf and libprintspool.

## 6 Future plans

There is a magic to open source development. For a developer, it is always a bit like stepping into the unknown and then allowing things to happen around you rather than forcing something to happen a certain way. Much of this is the way that I have conceived printing working. One of the things that I have always found interesting about the open source world is that projects take on a life of their own and as the little bits of technology that we build become more widely distributed and get incorporated into other pieces of technology they become mutated and evolve into something that is different and beyond what I ever conceived.