Linux Printing Tutorial at Linux−Kongress 2002 Cologne, Germany:

(V.) Dissecting The CUPS Filtering System:

A Network Postscript RIP For non−PS Printers

Workshop at Linux–Kongress 2002 Cologne, Germany:

Dissecting The CUPS Filtering System: A Network Postscript RIP For non–PS Printers

Contents

1. Printing via Windows NT print servers
   1.1. Client–side driver operation
   1.2. Server–side driver operation
2. Windows client printing via UNIX print servers
   2.1. Client–side drivers: raw print jobs
   2.2. Server–side "drivers": none
3. CUPS and Windows clients
   3.1. Client–side drivers: raw print jobs, just like in the past...
   3.2. Server–side "drivers": via client PostScript print files
   3.3. CUPS –– a PostScript RIP valid for 1.000++ non–PostScript printers
   3.4. PPDs to propagate print job options
4. The CUPS–internal filtering system: a PostScript RIP
   4.1. The pre–filters
   4.2. The *pstops*–filter
   4.3. The *pstoraster*–filter
   4.4. The *imagetoraster*–filter
   4.5. The *imagetops*–filter
   4.6. The printer–specific filters
   4.7. Foomatic and the *cupsomatic*–"filter"
   4.8. Overall picture: CUPS–filters with Foomatic, Gimp–Print & TurboPrint plugged–in
5. How it is controlled
   5.1. */etc/cups/mime.types* –– rules to determine MIME types
   5.2. */etc/cups/mime.convs* –– assignment of filters according to MIME types
6. Practical Applications
   6.1. Stabilizing Windows Terminal Servers
   6.2. Achieving centralized accounting
7. *cupsaddsmb* –– the easy way to share a CUPS driver to Windows clients
8. A few figures about needed resources
9. Summary

A. APPENDIX
   A.1. CUPS Web Interface
   A.2. CUPS Browsing
   A.3. CUPS Commandline Interface
   A.4. CUPS GUI (KDEPrint)

by Kurt Pfeifle,

Network Printing Consultant,
Danka Deutschland GmbH,

Author of CUPS–FAQ and KDEPrint Handbook

CUPS, the Common UNIX Printing System, is thought by many to become an epoch making printing application –– for the Linux and UNIX realms of computing. What is not yet clearly seen: it could also –– in tandem with Samba's new features of RPC–based printer driver downloads –– make a big impact on Windows clients network printing in the future.

This paper dissects the inner workings of the CUPS filtering system and envisages a few ideas about its deployment as a *print server of a new kind*: not just *spooling* Windows client print files as "raw" jobs, but also processing them through its internal PostScript RIP (*Raster Image Process*) –– thusly providing more stability and job accounting features to its environment.

Workshop at Linux–Kongress 2002 in Cologne, Germany:

Dissecting The CUPS Filtering System: A Network Postscript RIP For non–PS Printers

by Kurt Pfeifle <kpfeifle@danka.de>
Author of the CUPS–FAQ
and the KDEPrint Handbook

CUPS, the Common UNIX Printing System, is thought by many to become an epoch making printing application –– for the Linux and UNIX realms of computing. What is not yet clearly seen: it could also –– in tandem with Samba's new features of RPC–based printer driver downloads –– make a big impact on Windows clients network printing in the future.

This paper dissects the inner workings of the CUPS filtering system and envisages a few ideas about its deployment as a *print server of a new kind*: not just *spooling* Windows client print files as "raw" jobs, but also processing them through its internal PostScript RIP (*Raster Image Process*) –– thusly providing more stability and job accounting features to its environment.

Late–breaking news about Easy Software Products' (the developers of CUPS) license deal with Apple, to supply CUPS to Darwin and Mac OS X, have increased the likelihood of more people giving it a spin. Therefor more knowledge about its architecture is a dire need. CUPS is easy to deal with –– more easy than any other printing system. But it is also new –– and it is different from any of its predecessors. Therefor there ***are*** new things to learn and to understand as well... .

One of the first obstacles for newbie users and administrators of CUPS is: to come to grips with its internal filtering system.

Therefor we will spend a major part of this paper on this aspect, which then leads to some practical benefits: how to use a Samba/CUPS print server as a networked *Raster Image Processor* (RIP), first interpreting PostScript printfiles on behalf of its clients and then sending them off to their final print destinations. We will not deal much with other benefits of CUPS here such as its firm foundation in the new IETF standard *Internet Printing Protocol* (IPP), its complete support for print job options, its clustering and failover capabilities or its ingeniously simple and effective "printer browsing mechanism", allowing clients to "print without a driver" in a true Plug'n'Play manner.

1. Printing via Windows NT print servers

Suppose, a Windows NT client installs a printer. The jobs should be sent via an NT print server. Obviously the client needs a "driver" too. You may choose here, where the "driver" should be installed: either locally or on the print server.Do you have the same choice with UNIX print servers? Install a printer driver to run on the server, with the server processing the files? With the client having the full control over job settings, just as if the driver was installed locally? The answer used to be a plain "No!". With CUPS it's "Yes!". But let's first look at a "purely–Windows" environment...

1.1. Client side driver operation

Choosing the *local driver installation* for the NT client will process the printfile on the local machine. Basically, the driver will take the output of the Windows *GDI* (Graphical Device Interface), called an *Enhanced MetaFile* (EMF) and process it according to the target printer. A PCL driver will generate PCL output from the GDI input, a PostScript driver will output PostScript and so forth. Of course, print options (duplex, punch, staple...) are all available, as provided by the driver. The client will be able to send a printjob off his application, even if the server was temporarily unavailable (spooling the job locally in the meanwhile), but no other client will be able to utilize this driver.

1.2. Server side driver operation

Choosing the driver installation to take place *on the server* will behave a bit different. It will cause the printing client to send the output of his local GDI largely "as is" (EMF, *Enhanced MetaFile*) towards the server. The server–installed driver will in turn process it and generate the correct format (PostScript, PCL, Raster...) suitable for the target printer from the client's GDI output upon arrival of the file. Print options to select are available for the client here too; there is hardly any visible difference in the drivers' local GUI for the client as compared to a "direct" client side driver installation. However, the client will not be able to send a printjob off his applications if the server is temporarily unavailable. On the other hand, the server–side printer driver installation is available for other clients to use (provided it is "shared") through a simple "point and print" event.

2. Windows client printing via "classical" UNIX print servers

"Traditionally configured" UNIX print server have been available for Windows clients for a long time. They provide one same basic service as Windows print servers: to make sure that the printer is not bogged down by possibly too many concurrent access attempts (as could happen in networks with a purely "peer–to–peer" printing system), by providing a central spooling access point to the printer, that is "always open" to the clients.

2.1. Clients side drivers: "raw" print jobs

Classical UNIX print servers can't offer the *file conversion service* (server−side printer driver) to Windows clients like Windows NT servers do. Clients need to have a local driver installed, suitable for the target printer, and send their pre−formatted jobs as "raw", telling the server to pass it to the printer untouched. For a long time UNIX servers were even unable to offer printer drivers for download and semi−automatic installation.

2.2. Run server side "drivers": not possible!

UNIX print servers simply don't know how to convert their Windows clients' GDI output, as there are no such "drivers" or filters available for them. Therefor UNIX print servers will normally just play the role of a *central spooling device* for a lot of clients and printers, storing jobs temporarily and passing them on as the printer(s) become available again after they finished their current job(s).

There occurs not very much CPU− load for a spool−only server. It is common practice to use an otherwise shelved computer as the print server, because all it needs to have is enough hard disk space to take in the arriving jobs and hold them as long as needed. The processing power usually is no bottleneck for a "spool−only" UNIX print server. These setups are re−nowned for their stable and fast service in fulfillment of their duties, in most cases better than their Win NT counterparts (to be fair, however, this never involves the server side processing of printfiles with some major format conversion, as this needs to be done by the clients themselves).

2.3. Storing client drivers for download and automatic installation: yes! (And maybe more?)

However, with the arrival of the Version 2.2.0 of Samba, you can now comfortably *store* the drivers (even via easy upload from a client) on a UNIX/Samba print server, and offer them to WinNT/2K/XP clients for download and local semi−automatic installation. Due to the complicated nature of this process and the closed source of the involved MS−RPCs (*Microsoft Remote Procedure Calls*), it took until Version 2.2.4 to get this feature fairly stable and make it work for a lot of different environments. But now it's there −− and this is a huge achievement. It eases administration for large Samba driven client networks a lot. At the same time it is opening up new creative applications for other software, such as discussed here in conjunction with CUPS..

3. Windows client printing via a CUPS print server

The different puzzle pieces of Samba printing in the past only offered partial solutions to users and administrators. With the new RPC−based printing functions, Samba can provide a much better service now, even with traditional spoolers.

With CUPS you can arrive at a new peak. There is now a *convenient* way to provide server−side printjob processing for Windows clients by a CUPS server, similar to what it does in relation to its native CUPS clients.

This model includes some very tempting features to users and administrators:

- You have a fully−fledged "driver" *run* on the server, processing jobs for clients from all plattforms −− Linux, Windows, UNIX and Mac OS X in much the same way.
- All Windows clients can "install" this driver upon first contact with one simple click, using Samba's new "point and print" feature −− UNIX and Linux clients don't even need to do anything to get the same.

3.1. Client−side drivers: raw print jobs, just like in the past...

Of course, CUPS is not just viable if you need to have (or want) this "server side job processing". Just as with traditional LPD−based print servers, you can also set up CUPS to act as a *mere spooling host* for its Windows clients. This role it fulfills equally reliably as its outmoded counterparts.

3.2. Server−side "driver": processing client PostScript print files

The really new shot is to set up the CUPS print server as an application that processes the client jobs on the server side. Jobs then don't get rendered print−ready already on the *client side* (using the vendor printer driver there), but *on the server* before passing them on to their final print devices.

In difference to a Windows NT print server, this rendering, of course, is *not* based on GDI. CUPS still can't process any GDI−originated input. What's the deal then?

The secret lies in the use of PostScript. The idea is this: clients on the Windows platform use an Adobe PostScript driver, armed with a CUPS PPD to be able to control all job options. They send these print jobs to the CUPS server. The server takes care to process the files as intended, including the respecting of all set job options.

3.3. CUPS −− a PostScript RIP valid for 1.000++ non−PostScript printers

This is nothing new and revolutionary if the *printers themselves* understand PostScript: In this case this is a valid option for *all* spooling systems, not just with CUPS. But ***with CUPS, you can do this, even if your target printers (or some of them) don't use PostScript***. There is only one pre−condition: CUPS itself must be able to print natively onto the printer(s) in question. Given, that there are now more than 3.000 CUPS−driver/PPD−combos available, in several languages, from various sources, including some commercial ones, this is not a big deal.

This setup is a very tempting offer to satisfy at least two different requirements:

- the need to have *centralized accounting and billing* for all printjobs;
- the need to *remove any printer driver induced instabilities* stemming from a multitude of non−PostScript printer operating in f.e. a Windows Terminal Server environment

This way you'll have a network transparent PostScript RIP, able to deal with thousands of different printer models, based on Ghostscript. This way, CUPS lets appear all printers to the clients to be PostScript printers. It takes over the task to convert the PostScript into print−ready data on behalf of those printer(s) which don't take PostScript as their native input. CUPS becomes a *network PostScript RIP implemented in software*, capable to run thousands of models, with the full power of device features under the control of client users and administrators...

3.4. PPDs to propagate print job options

PPDs (*PostScript Printer Descriptions*) are a well known mechanism in relation to PostScript printers to fully control job options. They allow the user to flexibly change print setting according to his preferences.

PPDs are part and parcel of PostScript printer drivers. They are a de−facto standard, written in pure ASCII. They follow a standard layout and syntax format, defined and published by Adobe. They are also used to dynamically build a GUI driver interface that lets you select print options. Accompanying the user−visible options in human−readable form, they also contain PostScript, PJL (*Print Job Language*) or PCL (*Printer Control Language*) code representing the user options to the printer. These commands are sent to the printer, embedded in the PostScript code, following user selections.

CUPS is a fully PPD−aware and −literate printing system. It can use the PPD of any Windows NT PostScript driver to its full capacity. Those PPDs are written and distributed by the printer manufactures as part of their WinNT drivers. There is no need to write them yourself, if you use a PostScript printer..

But CUPS' PPD−support doesn't stop here. Its developers have ***extended the concept of PPDs to non−PostScript printers***. There is only one little decisive addition as compared to the "normal" PPDs: a line starting with the keyword "**cupsFilter*" tells the CUPS daemon on the server, which filter should be used to process the accompanying PostScript file. (Other applications coming across such a PPD−file will ignore this directive as it appears to them as a "comment"...)

CUPS−PPDs for non−PostScript−devices were still quite rare two years ago. Now there are several thousand avaliable, from different origins:

- *ESP PrintPro* (commercial, non−Free) is packaged with more than 3.000 PPDs, ready for successful usage "out of the box" on Linux, HP−UX, Solaris, IRIX, Tru64 and some more commercial Unices (it is written by the CUPS developers themselves and its sales help finance the further development of CUPS, as they feed their creators)
- the *Gimp−Print−Project* (GPL, Free Software) provides around 120 PPDs (supporting nearly 300 printers, many driven to photo quality output), to be used alongside the Gimp−Print CUPS filters;
- *TurboPrint* (Shareware, non−Free) supports roughly the same amount of printers in excellent quality;
- *OMNI* (LPGL, Free) is a package made by IBM, now containing support for more than 400 printers, stemming from the inheritance of IBM OS/2 KnowHow ported over to Linux (CUPS support is in a Beta−stage at present);
- *HPIJS* (BSD−style licences, Free) supports around 120 of HP's own printers and is also providing excellent print quality now;
- *Foomatic/cupsomatic* (LPGL, Free) from Linuxprinting.org are providing PPDs for practically every Ghostscript filter known to the world, now usable with CUPS.

More details on some differences between those PPDs (especially the Foomatic/cupsomatic way of doing things) is discussed in the following section.

4. The CUPS−internal filtering system

Thanks to its modular software design, CUPS is able to run more than 3.000 printer models, utilizing filters/drivers and PPDs from different origins. Hardware vendors or 3rd party software developers can easily plug their own drivers and filters into the CUPS framework, as is proofed by Foomatic, the Gimp−Print project or the TurboPrint shareware. Its inner mechanism consists of a number of individual filters which can be flexibly chained and concatenated to achieve the projected result.

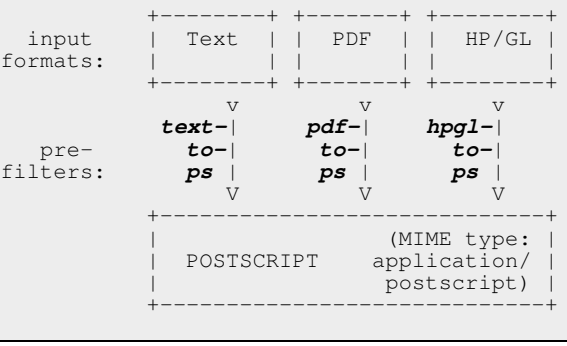
Let's have a closer look.

Upon receiving, print files are subjected to a "*MIME typisation*" by the CUPS daemon. (MIME types describe a file format in a non−ambiguous way. MIME types are diverted into different categories, with a scheme like *main_category/minor_category*. MIME types are registered with IANA, the *Internet Assigning Numbers Authority*. Vendors may register their own proprietary file formats with IANA provided they explain a method to recognize the format as such by a unambiguous method. CUPS has its MIME type recognition rules deposited in the (editable) file */etc/cups/mime.types*.

4.1. The pre−filters

Should there not arrive PostScript as an initial input format, there will be some pre−filters working on them. These pre−filter convert the input to PostScript (more exactly, into MIME type *application/postscript*). The filters have "speaking names": *texttops*, *pdftops*, *hpgltops*. (There is also *imagetops,* but this is, strictly speaking already more than a pre−filter*.):*

CUPS pre−filters generate PostScript from various input formats:



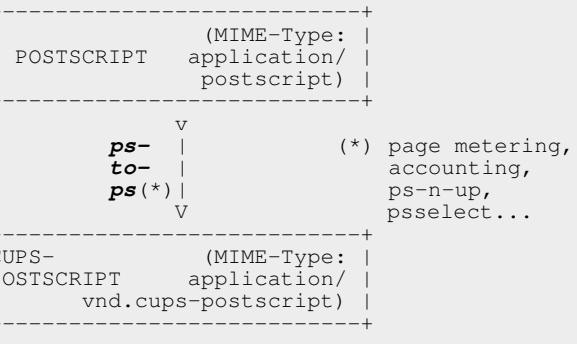
4.2 The *pstops*−filter

Once we have PostScript available (as MIME type *application/postscript*, which can as well be the initially supplied input format) the next filter in the chain is the *pstops* one.

Huh? Does this make sense? Generating PostScript output from PostScript input?

Sure. For this filter is doing a sort of "normalization" to the original PostScript. Additionally it is used to count the amount of pages processed, extract an optional page selection (a user might want to have on paper just the page numbers 4, 6–9, 11, 15−17 plus 13), or "impose" the document, by scaling and putting 2 or 4 pages on one sheet of paper. while its input is *application/postscript*, the output is *application/vnd.cups−postscript*, which indicates a "private" MIME−type vendor (= the CUPS developers) registration with IANA:

***pstops* counts pages, imposes multiple pages on one sheet or selects page−ranges for print:**

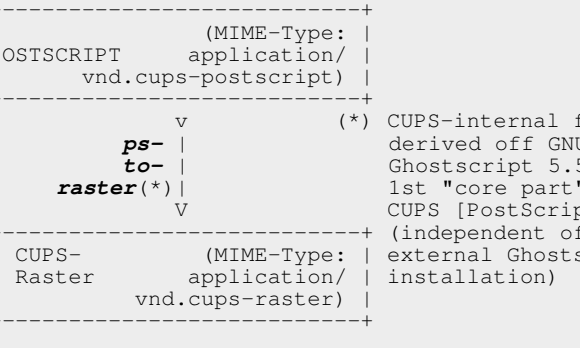


4.3. The *pstoraster*−filter

Next call is for *pstoraster*. This one is at the core of the CUPS RIP. Currently it is still derived off Ghostscript version 5.50. This sounds (and is) quite old, but the CUPS developers have applied many modifications and patches of their own to it an, so that the consecuting raster drivers in the chain still are able to drive many inkjet printers to true photo quality output. Gimp−Print, for instance, is praised by many of its users for producing better quality on Epson printer than the vendor−supplied Windows drivers! (Note: *cupsomatic* also taps in here, before pstoraster does its work, and "kidnaps" the CUPS−PostScript to channel it through the standard "classical" Ghostscript installation, before returning the resulting raster data back to a CUPs backend.)

pstoraster outputs raster data. This raster data is also registered with IANA as MIME type *application/vnd.cups−raster*. Its specification is of course open. The raster data describes the printfile as separate pages in the target printers's color space, with print options tagged inside separate page headers:

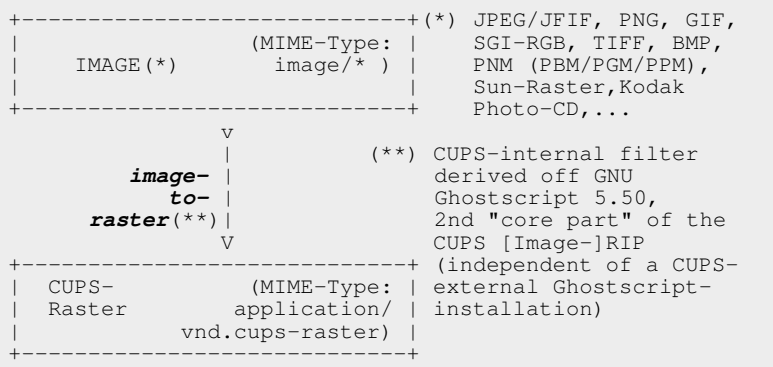
***pstoraster* outputs generic raster, foundation for post−processing by printer specific filters:**



4.4. The *imagetoraster*–filter

imagetoraster could here be on duty too, if the input file was of MIME type "*image/**", heading for non–PostScript printers. This has a similar role to pstoraster for all image input files, being the central CUPS *image file RIP*. It is used for all non–PostScript printers. Possible input MIME types are a rich variety of formats, spanning from JPEG/JFIF over PNG, GIF, TIFF, SGI–RGB, BMP, PNM (PBM/PGM/PPM), Sun–Raster or Kodak Photo–CD and more:

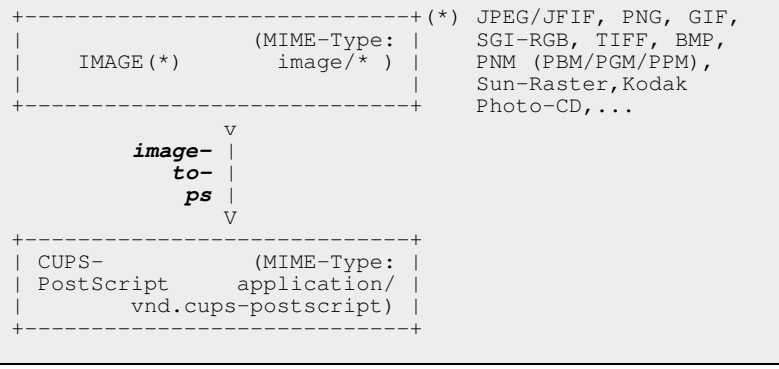
imgetoraster outputs generic raster, foundation for post–processing by printer specific filters:



4.5. The *imagetops*–filter

Alternatively, *imagetops* would do the job here, if the target printer was a PostScript–capable one:

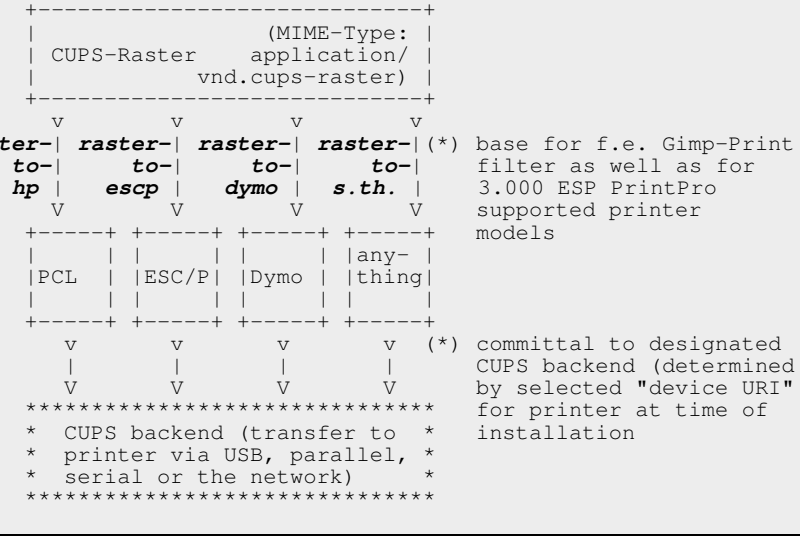
from image input, _imagetops_ outputs normalized CUPS–PostScript, ready to be passed on to real PostScript printers:



4.6. The printer–specific filters

Last, but not least, it's the turn to convert the generic CUPS raster data to a printer–specific format, embedding also all the control commands to get the desired print options from the device. This is done by printer–specific filters, or by calling a more general filter with printer–specific parameters. At this point, other deliberate filters or raster drivers (even proprietary ones) could be bolted onto the system, for example from vendors, who decided to have their models supported on Linux/UNIX or Mac OS X platforms.

printer specific filters generate printer specific raster data and from generic CUPS raster:



4.7. Foomatic and the *cupsomatic*–"filter"

How do "*Foomatic*" and *cupsomatic* fit into this picture?

Foomatic is the database of Linuxprinting.org. It contains the most complete info about printer models, eligible Ghostscript drivers, supported print options and "drive execution details". One can online a access all these info and even generate a current PPD for any supported printer/driver–combo, using the "*PPD–O–Matic*" script running on the web server. [***Note***: you shouldn't any longer use the PPDs generated by the old script called "*CUPS–O–Matic*", as these are incompatible with some applications).

The easiest way to do it: select your printer model, read the notes about it, choose the "recommended driver" (sometimes there are as many as 5 drivers available, but there is always a recommended one), click on "Generate PPD", save the resulting PPD file and follow the instructions to install it. PPD–O–Matic–generated PPDs may not only be used with CUPS. They are even digestible for all PPD–aware programs (such as. StarOffice, OpenOffice, WordPerfect or CorelDraw for Linux). The PPD contains all available info about the driver/printer combination. PPD–aware programs can load the PPD to read stuff like needed page margins, available printer fonts from them or learn which PostScript code snippet needs to go into the printfile, if page 5 is to be drawn from paper tray 3, while page 9 needs tray 1, while the rest is from tray 5 (StarOffice).

You can even install those PPDs on any Windows client on top of an Adobe PostScript driver. PostScript print files generated with this combo can then be sent to a CUPS server with the fitting printer installed, which will process the file and let it print.

For the PPD to actually work on the CUPS server, you need to have a few more pre–conditions met:

- you need to have the *cupsomatic* Perl script installed in */usr/lib/cups/filters/*; this mimics to be a CUPS filter and is called from inside the PPD–O–Matic–PPD, but then it hands over the PostScript to an appropriate filter from your "standard" Ghostscript installation;
- you need to have the above mentioned *Ghostscript filter* (or "*DEVICE*") installed, which is called by cupsomatic;
- you need to have some *Perl modules* installed that are needed by cupsomatic to run correctly.

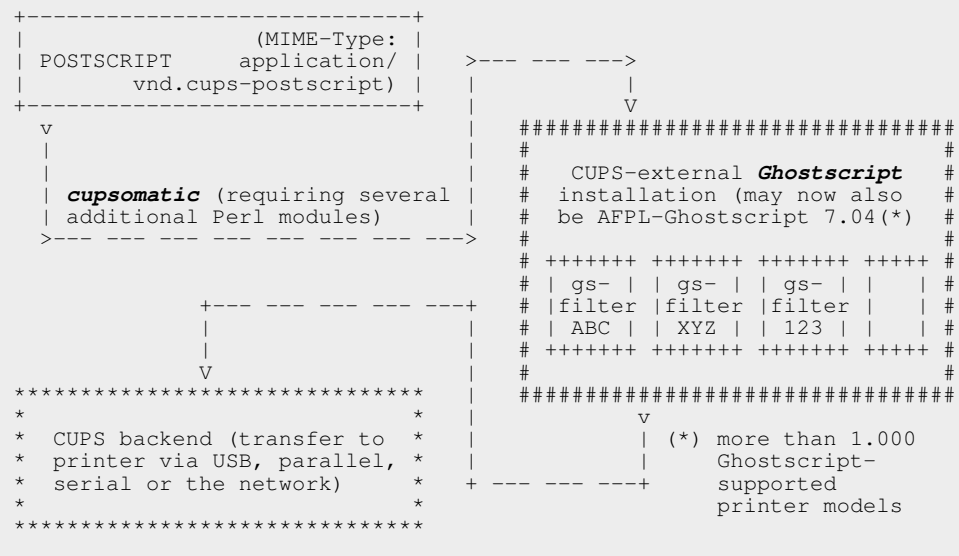
Similar to any other "non–PS–printer–PPD", the PPD–O–Matic generated ones contain a line starting with "*cupsFilter":

```
*cupsFilter:"application/vnd.cups-postscript 0 cupsomatic"
```

This tells CUPS to use *cupsomatic* as the next filter after it has reached the stage of a MIME type of "*application/vnd.cups–postscript*" through its standard filtering mechanism. cupsomatic then reads the attached PPD options and constructs from them a correct Ghostscript commandline to run the file through the appropriate filter. In the end, it hands the converted file back to a CUPS backend for sending it off to the printer.

This way Foomatic/cupsomatic make all working combinations of printers and Ghostscript filters known to mankind also available for use in CUPS. Of course this produces basically the same quality as other Ghostscript–driven spoolers do for their printers –– be it bad (sometimes) or average (more often) or excellent (rising).

***cupsomatic* "kidnapping" CUPS–PostScript, to let it be filtered and converted by Ghostscript:**

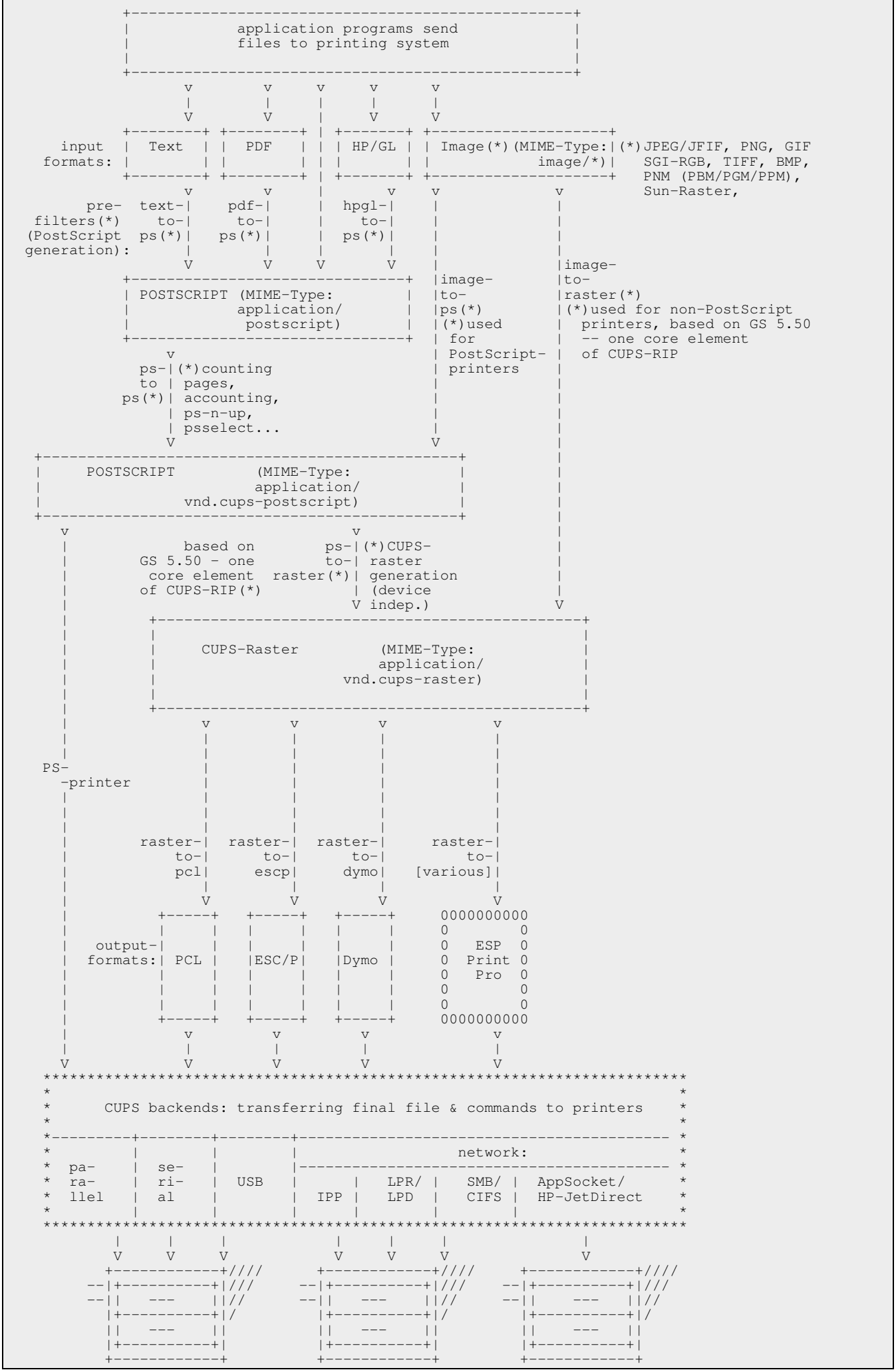


4.8. Overall picture: CUPS–filters with ESP PrintPro, Foomatic, Gimp–Print and TurboPrint plugged in

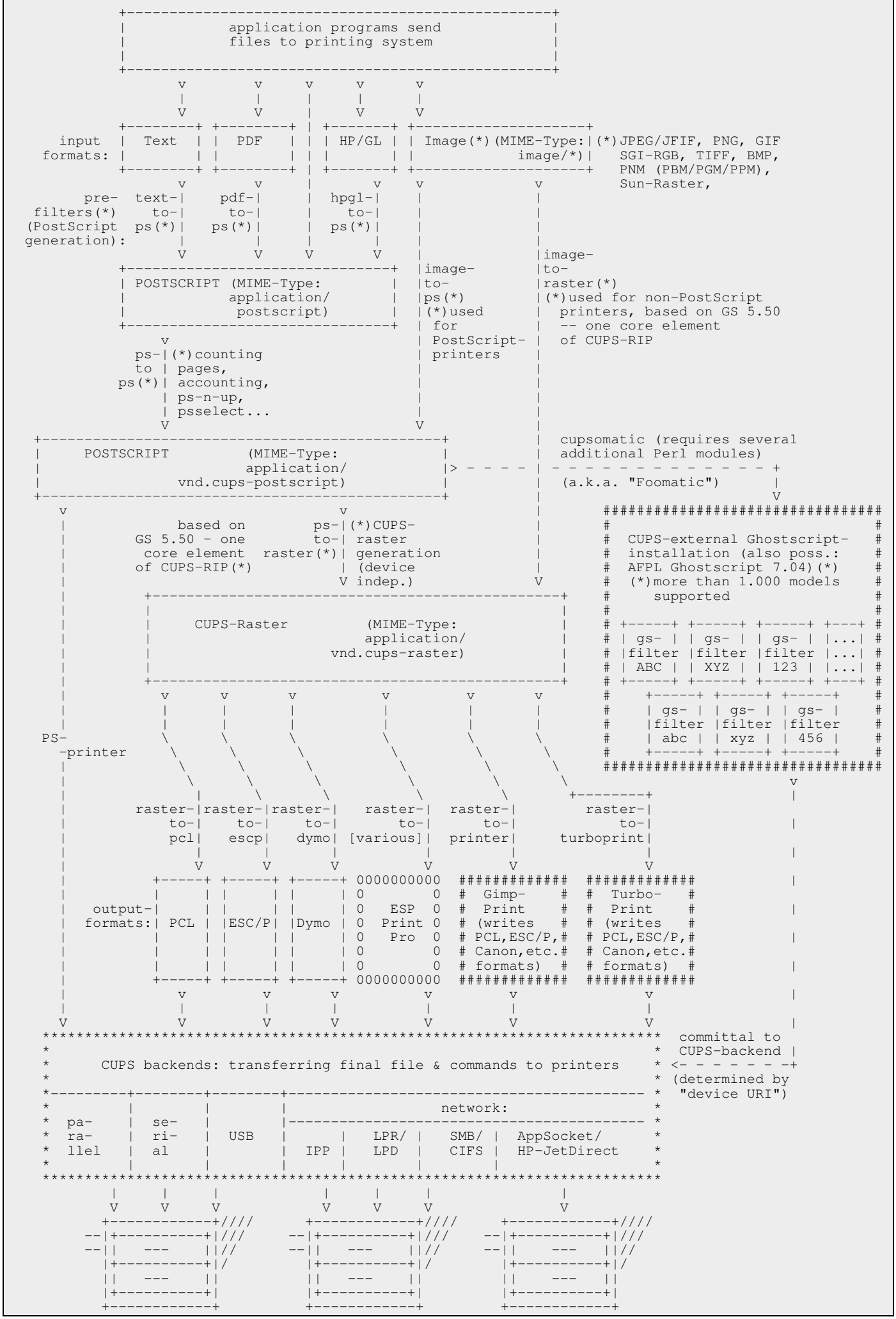
A complete overview on the CUPS filtering chain is drawn in the following two charts.

- The first chart includes the professional CUPS extension ESP PrintPro, which provides largely improved versions of pstoraster and imagetoraster, plus more than 3.000 printer drivers working "out of the box", and a GUI of its own.
- The second chart shows, were any of the available "3rd party add–ons" to CUPS fit in. They include Gimp–Print, TurboPrint and Foomatic.

Overall picture for CUPS filters, with plugged-in *cupsomatic* (from Linuxprinting.org), Gimp-Print & TurboPrint:



Overall picture for CUPS filters, with plugged–in *cupsomatic* (from Linuxprinting.org), Gimp–Print & TurboPrint:



5. How it is controlled

There are two important files which determine the guidance of printfiles through the CUPS filtering chain.

5.1. */etc/cups/mime.types* –– rules to determine MIME types

In */etc/cups/mime.types* there are the "rules" contained which determine the MIME types of any given file and also which ones are recognized by CUPS.

5.2. */etc/cups/mime.convs* –– assignment of filters according to MIME types

In */etc/cups/mime.convs* the filters to be used for any given MIME type are assigned. CUPS tries to construct a valid filter chain from the beginning to the end, also under the directive of the PPD. Every filter also has a virtual "cost" attached to it. In case there are different possible solutions CUPS takes the one with the "cheapest" sum.

6. Practical applications

To implement a CUPS server as a networked software RIP based on Ghostscript, has several benefits.

6.1. Stabilizing Windows Terminal Servers

Windows Terminal Server (WTS) are notorious for their problems regarding printing. These servers often run MS Office and other applications for dozens of clients (whose workstations serve merely as a TV set providing a "window into the server"). A WTS serving 40 users with a small desktop printer each needs to also run 40 printer drivers (or at least one for each different model). It is estimated that 50 % of Blue Screens are related to printer drivers problems. Many sites have adopted a policy to just allow one PostScript and one basic PCL driver for all models of devices in their realm, thusly reducing their users to merely print "simplex loose sheet collections " (while their hardware could do a lot better, of course).

Here's a possible solution:

- relieve the WTS from their many printer drivers;
- consolidate on an Adobe PostScript driver (not known to cause any problems);
- make the WTS a CUPS print client, using the PS driver;
- empower the PS driver on the WTS with multiple CUPS–PPDs;
- increase WTS uptime again;
- make users happy and give them back the ability to use all functions of their printers.

6.2. Achieving centralized accounting

CUPS maintains a *page_log* in */var/log/cups/*. In there, every processed job is logged, with every page printed, number of copies, time of the day, username, job id, target printer and optional billing information. A typical file looks like this (extract):

***/var/log/cups/page_log* (enumerating printername, username, job–id, time, page–no., no of copies and billing info:**

```
lj5gray_foomatic kde4 87 [27/Apr/2002:01:58:05 +0200] 1 4 Sylvia
lj5gray_foomatic kde4 87 [27/Apr/2002:01:58:05 +0200] 2 4 Sylvia
lj5gray_foomatic kde4 87 [27/Apr/2002:01:58:05 +0200] 3 4 Sylvia
lj5gray_foomatic kde4 87 [27/Apr/2002:01:58:05 +0200] 4 4 Sylvia
lj5gray_foomatic kde4 87 [27/Apr/2002:01:58:05 +0200] 5 4 Sylvia
lj5gray_foomatic kde4 87 [27/Apr/2002:01:58:05 +0200] 6 4 Sylvia
lj5gray_foomatic root 88 [27/Apr/2002:02:04:52 +0200] 1 1
lj5gray_foomatic root 88 [27/Apr/2002:02:04:52 +0200] 2 1
lj5gray_foomatic root 88 [27/Apr/2002:02:04:52 +0200] 3 1
mopi_320_mailbox kurt 89 [27/Apr/2002:16:23:15 +0200] 1 1 marketing
mopi_320_mailbox kurt 89 [27/Apr/2002:16:23:15 +0200] 2 1 marketing
```

NOTE: This information is collected and logged while the printjob passes the *pstops* filter. It therefor contains the *"RIP–ped"* pages. It may possibly not reflect the actual *printed* pages: if the job gets lost *after* the RIP–ping finished, on the way to the printer, or inside the printer. But overall, it is already a good tool to track costs.

This log file contains all data to generate business reports from. It may also be written to a different place (syslog).

If the clients print "raw", CUPS logs the job as containing 1 (one) page. The logfile content in these cases also doesn't reflect the actual printed pages. "Raw" jobs are *bypassing* the pstops filter, which is doing the tallying at the current stage. During the 1.2.x development cycle of CUPS this accounting task will be assigned to the CUPS backends, talking directly to the printers and querying and logging each sheet as it is printed. These backchannel data will provide much more accuracy and generally also improve feedback to the user.

This information contains the "RIP–ped" pages. It may be incorrect, if the job gets lost on the way to the printer, after the RIP–ping finished. But overall, it is already a good tool to track costs.

7. *cupsaddsmb* –– the easy way to share a CUPS driver to Windows clients

CUPS provides a nice utility to share any given CUPS printer to Windows clients: *cupsaddsmb*. If you call it with the "–a" parameter, it shares all printers (i.e. it copies the printers' PPDs plus the Adobe driver files to the [print$] share.:

```
cupsaddsmb  -U root -v -a
```

(*–v* is for verbose output, *–U* is to name a user with the access right to do his). To share an individual printer do

```
cupsaddsmb  -U root -v  my_printer_name
```

Of course, you need to prepare the *[print$]* share (holding the downloadable Adobe PostScript drivers) beforehand. This amounts to change the *smb.conf* to contain the following entries:

```
[global]
   load printers = yes
   printing = cups
   printcap name = cups

[printers]
   comment = All Printers
   path = /var/spool/samba
   browseable = no
   public = yes
   guest ok = yes
   writable = no
   printable = yes
   printer admin = root

[print$]
   comment = Printer Drivers
   path = /etc/samba/drivers
   browseable = yes
   guest ok = no
   read only = yes
   write list = root
```

In addition you need to download the Adobe Postscript driver files from their website. Once you have extracted the driver files, create a "*drivers*" directory in the CUPS data directory (usually */usr/share/cups/*) and copy the Adobe files using UPPERCASE filenames, as follows:

```
ADFONTS.MFM
ADOBEPS4.DRV
ADOBEPS4.HLP
ADOBEPS5.DLL
ADOBEPSU.DLL
ADOBEPSU.HLP
DEFPRTR2.PPD
ICONLIB.DLL
```

Users who bought a ESP PrintPro license have no such work at all. They just tick the "Install Samba driver files" inside the installer window and they are done. "*cupsaddsmb*" can then also be called from a GUI, by clicking on "*Export Printer*".

8. A few figures about needed resources

Of course, as was hinted to earlier, making CUPS to act as a Software RIP on behalf of its clients and printers, consumes much more CPU than just mere spooling and passing on the jobs. Here are a few figures to help you estimate the size of the machine you'd need. Assume the following facts:

Assumptions about daily print runs:

- you run a network with 100 printers;
- you want to guarantee a maximum of 10 printjobs to be concurrently processed;
- you have an average of 50 printjobs/day for each printer (5.000 jobs daily total);
- a typical job size is up to 20 MByte:

First, assume your printers are all PostScript capable. This requires the following computing capacity:on your CUPS server

100 PostScript printers, 5.000 jobs/day total, 10 concurrent jobs guaranteed, typical job size 20 MByte:

- 1 x 400 MHz Intel processor(s) or 1 x 200 MHz non–Intel processor(s)
- 98 MB RAM (including 64MB of operating system overhead)
- 5.9 GB Disk (including 2GB of operating system overhead)

To enforce the requested limits, set these CUPS configuration directives (in */etc/cups/cupsd.conf*)

- FilterLimit 1.000
- MaxJobs 1.000

You need more computing power, if you assume all the printers to be non–PostScript models (other assumptions being equal):

100 non−PostScript printers, 5.000 jobs/day total, 10 concurrent jobs guaranteed, typical job size 20 MByte:

- 2 x 800 MHz Intel processor(s) or 2 x 400 MHz non−Intel processor(s)
- 188 MB RAM (including 64MB of operating system overhead)
- 5.9 GB Disk (including 2GB of operating system overhead)

To enforce the requested limits, set these CUPS configuration directives (in */etc/cups/cupsd.conf*):

- FilterLimit 2.500
- MaxJobs 1.000

---

9. Summary

CUPS and Samba can make up a great tandem to serve Windows client networks with unique printing services:

- Loading down printer drivers to the clients via "Point and Print" eases administration a lot.
- Running CUPS as a network software RIP centralizes and eases accounting and billing considerably.
- Consolidating Windows Terminal Servers by reducing them to run a single PostScript printer driver type for multiple clients (instead of many different ones), stabilizes their performance.

CUPS is posed to gain a much more important role in the coming years.

Stuttgart, April, 28th, 2002

Appendix

Here are some basic infos about CUPS, dealing with three possible interfaces to its functions: web interface, commandline and GUI. Basics about the "CUPS printer browsing" are also outlined.

The CUPS web interface: How does it look like?

A lot of views and actions upon the CUPS server are available through its web interface. This also works via the IPP *well known port* 631. If you point your browser to *http://localhost:631/* you arrive at the starting page, with links to the CUPS online help documents, printer and job lists and an administration page (this one allows addition, deletion, modification, configuration and testing of printers and printer classes).

NOTE: After a default CUPS installation, there is no immediate *remote* access to this web interface (only from *localhost*). Refer to the CUPS documentation to find out how to open it for remote access. (HINT: it is in *cupsd.conf*, settings are done by *Allow From...* directives in the appropriate *Locations* for *admin*, *printers* and *server root*.). Here are a two screenshots.

Left: default printer options are set via the "Configure Printer..." button. Right: list of available printers.

CUPS−Browsing: How does it work? What are the benefits?

CUPS includes an extension to IPP, called the "CUPS browsing". It enables CUPS clients to "discover" available printers. Further, it allows clients to "just print it" without further configuration of printers, queues, drivers etc. How does this work?

In "CUPS−speak", all machines that send jobs directly to printers are called "servers". The queue could be a "raw" queue, having no driver and PPD associated with it. Most printers will have a PPD mapped to them (which indirectly also includes the mapping to a CUPS−Raster− or a Foomatic−Ghostscript−driver). All printers on a CUPS server are kept in the file */etc/cups/printers.conf*. This files contains entries of this format:

from */etc/cups/printers.conf* (an entry describing one printerwith no Location− or other Info−Strings):

```
<Printer DANKA_Color_50ppm>
  DeviceURI lpd://10.160.31.85/direct
  State Idle
  Accepting Yes
  JobSheets none none
  QuotaPeriod 0
  PageLimit 0
  KLimit 0
</Printer>
```

This is one of the shorter entries. It describes a printer with its name *DANKA_Color_50ppm*, connected with a device URI of *lpd://10.160.31.85/direct*. Next is a longer entry. It describes a similar printer, using a banner page called "*Confidential*" at the beginning of each job. It also includes a printer info and the printer location (as would appear in the web interface or in a GUI) and the quota details enabled for that printer, and finally the special access control for it. Don't worry, you are not meant to create the printers.conf file manually. It is written by CUPS depending on what you tell it with the *lpadmin* command and it is also kept up to date by CUPS. Not all entries need to be there for every printer, as shown in the previous shorter example:

from */etc/cups/printers.conf* (an entry describing one printer including Location− and other Info−Strings):

```
<Printer DANKA_hispeed_color_50ppm>
  Info Printer does best Laser Color Prints known on market
  Location Training Center Networkprinting/Samba/IPP/CUPS/ESP PrintPro
  DeviceURI socket://10.160.31.85:9100/
  State Idle
  Accepting Yes
  JobSheets Confidential none
  QuotaPeriod 604800
  PageLimit 100
  KLimit 5000
  AllowUser kurt
  AllowUser root
  AllowUser chef
</Printer>
```

If a CUPS machine shall become a **server** for other clients and "share" its printers, you need to specify a "*BrowseAddress*" in c*upsd.conf*. This causes CUPS to announce its printer list (basically the contents of its *printers.conf* file) via regular UDP broadcasts through port 631 to the address specified. Example: a *BrowseAddress 10.160.31.255/255.255.240.0* tells all nodes in the IP range of *10.160.{16−31}.[.0−255]* about his printers (in this example subnet mask is 255.255.240.0 for the LAN concerned). You can safely estimate an average of around 150 Bytes for the amount of data per printer and per broadcast occurance (of course, this is largely dependent on the length of the strings related to "*Location*" and *"Info*" which you gave during printer installation). Compare this to the noise on the wire two NT boxen create while fighting their master browser elections! So much for the bandwidth consumed by CUPS printer broadcasts....

Clients are *listening* for server broadcasts arriving at port 631. From this info they build their list of available printers and cache it for a certain time (defaults to 5 minutes, to allow the bridging of individual update broadcast not coming through). That's all they need for printing....

Wait! What about their drivers? How do they get the printer options info? Isn't there also broadcasting of all the printer PPDs involved?

No. Of course the clients can use printer options. And of course they need to know about available options beforehand. How can they retrieve them? This is discussed in the next two sections about commandline and GUI interfaces to CUPS print options.

The CUPS commandline interface: How does it look like?

From the commandline you can use most BSD–LPD or System V print commands with much the same options: *lp*, *lpr*, *lprm*, *lpc*, *lpq*, *lpadmin*... But remember, they are *new* implementations for those traditional commands. While most of their commandline parameters are the same (for backward compatibility), it is no mistake to consult the man page for the command to get clarity.

One of the most interesting new commands is the "*lpoptions*" command. It works across the network. You can ask a remote CUPS server about print options for a certain printer. Look at this (shortened) example:

Querying for long option list of a printer on remote CUPS server (returning PPD info related to this printer):

```
kde4@kde-bitshop:~/CUPS-Apple> lpoptions -h charlie -d LaserJet8100 -l
   Duplex/Duplex: *DuplexNoTumble DuplexTumble None
   PageSize/Medien-Größe : [....] A3 *A4 A5 B4 B5 [....]
   InputSlot/Medien-Quelle : [....] Upper ManualFeed *Middle Lower LargeCapacity Tray5 [....]
   OutputBin/Output Destination: OutputBin1 *OutputBin2 OutputBin3 [....] Stacker Stapler Upper
   PageRegion/PageRegion: [....] A3 A4 A5 B4 B5 [....]
   Option20/Accessory Output Bins: MBM5S *MBM7 MBM8 Standard
   Option3/Duplex Unit: *True False
   Option5/Envelope Feeder: True *False
   Option21/Multi-Bin Mailbox Mode: [....] MailboxModeStacker *MailboxModeMailbox
   Option4/Printer Hard Disk: *True False
   InstalledMemory/Total Printer Memory: 16-19MB 20-23MB 24-27MB 28-35MB *36MB
   Option2/Tray 4: *True False
   JCLEconomode/EconoMode: *False True
   JCLHoldKey/PIN (for Private Job): HoldKey0067 HoldKey0089  [....] *HoldKeyNone
   JCLResolution/Resolution: 300dpi *600dpi
   JCLUser/User: User1 User2 User3 User4 User5 User6 [....] User19 User20 *UserSystem
   HPCollate/Collate: False *True
   HPPaperPolicy/Fit to Page: *A4 Letter NearestSizeNoAdjust NearestSizeAdjust PromptUser
   HPHalftone/Levels of Gray: *Enhanced PrinterDefault Standard
   HPwmLocation/Print Watermark: True *False
   Smoothing/Resolution Enhancement: False *True
   HPwmText/Watermark: [....] Confidential Copy [....] *None Preliminary Proof [....]
   HPwmTextAngle/Watermark Angle: DegN15 DegN30 [....] *Deg60 Deg75 Deg90
   HPwmFont/Watermark Font: CourierB *HelveticaB TimesB
   HPwmFontSize/Watermark Size: pt24 pt36 *pt48 pt60 pt72 pt84
   HPwmTextStyle/Watermark Style: *Medium Narrow Halo Wide
```

Here we asked a remote host ("*–h*") named *charlie* to provide a long ("*–l*") list of print options for his printer *Laserjet8100*. What we get is the relevant extract of the *LaserJet8100.ppd* of this server, as is sitting in /etc/cups/ppd/. (We could also, to compare it with the commandline output of *lpoptions*, view the complete PPD by browsing to *http://charlie:631/printers/LaserJet8100.ppd*).

How do we interpret and use this output? For every option there is a separate line. Every line has an entry at the front (before the colon ":", consisting of two different parts, separated by a slash character ("/"). The first part is the internal name, also to be used if we later want to specify the option on the commandline. The second part is a string as will appear in the driver's GUI (or in the CUPS web interface if we click on the link to "*Configure Printer*"). (This is also the place were the localization of PPDs takes place. There is only one language supported per PPD.) Finally, after the colon, the values are enumerated, which are available for the option. There is always one option marked with an "*asterisk*"/star ("*") –– this is the currently active (= default) value for the option, used if no other value is specified by the user.

To *use a print command* with this printer *once* which puts a "watermark" named *Confidential* on the first page (but none on the others), you'd need to use this command:

Print command constructed with the info from previous option query:

```
lp -d LaserJet8100                \
          -o HPwmText=Confidential \
          -o HPwmLocation=False    \  # not obvious:"True" would
          -o HPwmTextStyle=Wide    \  # print WaterMark on all
          -o HPwmFontSize=pt42     \  # pages
          -o HPwmFont=TimesB       \
          -o HPwmTextAngle=Deg60   \
          /path/to/printfile
```

CUPS supports the creation of "*instances*" of printers. Think of an instance as of a fixed set of job options under a separate name, similar to a "*profile*". All instances of a printer belong to the same queue.

To set up this printer with a permanent instance named "*watermark*" (that puts the watermark named *TopSecret* on the first page, but none on the others, you'd need to use the "*lpoptions*" command like this:

Creating a printer instance for permanent and easy usage of the same setting:

```
lpoptions -d LaserJet8100/watermark \
          -o HPwmText=TopSecret     \
          -o HPwmLocation=False     \ # not obvious:"True" would
          -o HPwmTextStyle=Wide     \ # print WaterMark on all
```

```
            -o HPwmFontSize=pt42      \ # pages
            -o HPwmFont=TimesB        \
            -o HPwmTextAngle=Deg60
```

There is another utility named "*lphelp*" (written by Till Kamppeter), which can also query for CUPS printer options. *lphelp* is slightly different to *lpoptions*, with Pros and Cons:

- *Advantage no. 1*: You can use it to investigate a given PPD for the contained options (when the printer is not yet installed, and you want to find out if it is the correct one).
- *Advantage no. 2*: Its output is much more nicely formatted, and there is an explanation about the usage of the results.
- *Disadvantage*: It does not work across the network, only on localhost.

Here is a sample output for *lphelp*, aimed at the same printer:

Querying a printer for available print options with the assistance of *lphelp*:

```
kurt@transmeta:~> lphelp LaserJet_8100
    ============================================================
    HP LaserJet 8100 Series
    ============================================================
    Black & white printer
    Printer-specific options
    -------------------------

    Besides the options described in the CUPS software users
    manual (http://localhost:631/sum.html) you can use also
    the following options when you print on this printer with
    the "lp" or "lpr" command (a choice with the "default"
    mark represents the behaviour of the printer when the
    appropriate option is not given on the command line):

    Duplex:  -o Duplex=<choice>
    <choice> can be one of the following:
       DuplexNoTumble  (Flip on Long Edge (Standard), default)
       DuplexTumble  (Flip on Short Edge)
       None  (Off (1-Sided))

    Media Size :  -o PageSize=<choice>
    <choice> can be one of the following:
       A3  (A3, size: 11.69x16.54in)
       A4  (A4, size: 8.26x11.69in, default)
       A5  (A5, size: 5.83x8.26in)
    [............]
```

CUPS−GUI frontends: some screenshots

These commandline options, while invaluable for all those UNIX gurus out there, are not what desktop users dream about (other than during nightmares ;−). Fear not −− there also exist now a few GUI frontends to CUPS. The most developed one is undoubtedly the KDE version. KDEPrint has a utility called *kprinter*, which can be used from any non−KDE−application too, if this provides for a configurable "*print command*". The print command normally is "*lpr*" or "*lp*". Just replace the *lpr* or *lp* occurrences with *kprinter*, and you'll get a GUI to select your print options from. More info is found at http://printing.kde.org/. KDE's printing frontend may be used from StarOffice, Netscape, Mozilla, Galeon, Acrobat Reader, gv, any GNOME application and a lot more... Here are a few screenshots:

KDE−3.0, very complete GUI frontend for CUPS: "kprinter" replaces simple lp or lpr commands of traditional UNIX printing...

You can access most settings by clicking on the "*Properties...*" button on the initial dialog window. Settings may be saved as defaults. Different users may have different defaults. The "Driver Settings" tab is printer−specific. Its content is created from the PPD. In the following shot you see entries marked with a red color. These are "conflicting" entries. (Here, you can't select "Punching" if no "Finisher" is installed on the device). You don't get a warning if you choose conflicting options from the command line. The GUI is "safer" here. Tabs other than the "Driver Settings" are features provided by CUPS.

Left: "Driver Settings" (reflecting PPD contents); center: "General Settings"; right: "Image Settings" (for all image MIME types).

Acknowledgement & Credits (please read, even if you don't like it... ;–)

People whom I want to say "Thank You!":

- ***Jean–Eric Cuendet*** **for starting kups and qtcups, the predecessors of KDEPrint;**
- ***Michael Goffioul*** **for doing all the hard work recently;**
- ***Martin Konold*** **for thinking twice;**
- ***Sven Guckes*** **for teaching me essentials on the art of "survival on the terminal", just in case KDE is not there ;–) ;**
- ***Lars Mueller*** **(SuSE Linux) for building Samba and other packages to ease my testing and experiments;**
- **my employer *Danka* for giving me a chance to proof there is "a market for Unix print solutions" for them;**
- **...too numerous others to mention who let me snatch bits and bytes of knowledge off them and last, but not least:**
- ***Tom Schwaller*** **for encouraging me to get into "documentation writing".**

##########

Advertisement (don't read if you don't like it ;–)

Links and more info:

http://www.danka.de/printpro/faq.html
CUPS–FAQ by Kurt Pfeifle, more than 150 Questions (and presently a bit less Answers ;–)
http://www.pwg.org/ipp/
First hand infos about Internet Printing Protocol
http://www.cups.org/software.html
CUPS download
http://www.cups.org/documentation.html
First hand infos about CUPS
http://www.easysw.com/software.html
ESP PrintPro download
http://www.easysw.com/documentation.html
First hand infos about ESP PrintPro
http://www.linuxprinting.org/
Grant Taylor and Till Kamppeter: Linux Printing HOWTO, Linux Printing Database, PPD–O–Matic and PDQ–O–Matic and much more
http://www.pwg.org/ipp/IPP–Products.html
List of IPP–aware products at the Printer Working Group website