Table of Contents

CUPS:
Unix−Printing made easy −−
for Home Users and Network Administrators alike

by Kurt Pfeifle,
http://www.danka.de/printpro/faq.html
kpfeifle.at.danka.de

[Version 0.8.0]

Imagine a friend visiting. He has his Linux laptop with him, because you want to work on a common project. You have him quickly hooked up to your home network. Some time later, he needs to "quickly print a page"... and it just works! But he had never−ever seen your new super−duper 6−ink photo color inkjet. He doesn't need to install a driver first. And he still can "ring all the bells and blow all the whistles" of your device. He uses his own laptop to send the job and he can change the gamma value, the resolution, the saturation or the amount of Cyan used. −− An utopia? No. It's CUPS.

Printing Woes Cured

Clearly in the field of printing, especially when trying to use all the features of modern office printers, Linux and Unix were way behind other operating systems. Flexible setup of print job options like drawing sheets at will from different trays (or duplexing, punching, stapling and even creating pamphlets including the complete inline−finishing controlled by the client driver with center−folded, edge−trimmed and saddle−stitched staples on the final product), for a long time was just not possible.

Since two years a new printing software is spreading around the world, slowly first, now at an ever accelerating pace. CUPS, the *Common UNIX Printing System* is released under the GPL (*GNU General Public License*). It found its first friends amongst daring and curious home users. Many users succeeded the first time in their Linux career with the help of CUPS to produce reasonable or even photo quality output from their inkjets.

Fit for Enterprise Level Network Printing

What is hardly known even to happy CUPS users: CUPS is not only a supreme choice for single user environments. Its main strength lies in its unique features over the network.

This is now also proofed by the KDE project which is fully utilizing the power of CUPS in its new KDEPrint framework. With KDEPrint you can now construct and command an elaborate enterprise level network printing system. Features include complete administrative control over multiple print queues and their queued jobs, drag'n'drop printing, print job scheduling, print job priorization, billing and accounting support, print queue migration with possible adaptation of the print settings to the new printer's capabilities, per−printer print quotas, per−user print authorization and printer list filtering for restricting the number of printers which are listed in the users' printer selection dialog.

Magic Plug'n'Play Printing for Clients

CUPS clients can magically print even if they don't have any "driver" installed locally. They just plug into the network and wait for about 30 seconds. After that time, all printers are announced. A knowledgable power−user may then specify all print options even from the command line. And there is no single job setup feature missing! Others prefer to go to the web interface to set up the printer and send it then. Most comfortable however is the use of one of the GUI frontends, showing all print options "at your finger tips". Once you unplug the box from the LAN, it only takes 5 minutes (configurable) and the client has "forgotten" the printers and their options. Plugged into the next LAN, the roaming user is print−ready again after 30 seconds... Clearly this is a huge advantage CUPS provides for the Linux and various Unix platforms over those rival operating systems from Redmont or Cupertino.

GPL and Commercially Licensed

Wait... did I say "Apple"? Well, it is not true any more for Apple. Mac OS X from version 10.2 and Darwin will have CUPS built−in. At the time of writing this paper (Jan. 2002), this is not yet public knowledge, but the company is likely to announce this step on their Developer Conference beginning of May. By the time of reading this paper, it is probably an old hat for you... Apple will be licensing CUPS under a contract which exempts it from the rules of the GPL, while using and developing it under one of the Apple OSes.

The CUPS developers (*Easy Software Products*, a small software developing company, employing 3 people) distribute their product as a commercially supported and licensed software too. This is including CUPS, but under the name of "ESP PrintPro", with some substantial enhancements. ESP PrintPro is available for most common Unix platforms and ships with 2.500 printer drivers working "out−of−the−box". License and support contract income from these sales feed the developers and finance the further development of the GPL version of CUPS.

What's New with CUPS?

CUPS by far surpasses the existing standards for network printing: it is a complete replacement for the *Line Printer Daemon* (LPD as described by RFC 1179). CUPS puts the newly emerging industry and IETF standard *Internet Printing Protocol* (IPP) in control. IPP version 1.1 has been awarded the status of a "proposed standard" by the *Internet Engineering Task Force* (IETF). IPP had been drafted by the *Printer Working Group* (PWG, a loose association of printer manufacturers with the attendance of software companies such as Apple, Hewlett−Packard, IBM, Microsoft, Novell and SUN).

IPP provides functions, which unify printing across platforms and cure some decisive central weaknesses and fill the gaps of LPD:

- Authentication of users via passwords or certificates;
- Encryption of print data during transfer in between hosts or towards the printer;
- Advertisement of available printers on the local net to all potential clients.

But IPP doesn't invent the wheel for a second time. It basis itself firmly onto a proofen and robust data transfer protocol: HTTP 1.1. Of course, the PWG added some printing−related extensions to HTTP 1.1 (which is mainly providing the *transport*, whereas IPP is defining the *encoding* of printer and job attributes). But all the advantages of an established standard are preserved in IPP: you can easily plug−in TLS (*Transport Layer Security*) or SSL3 (*Secure Socket Layer, vers. 3*) for encryption purposes, LDAP (*Lightweight Directory Access Protocol*) or SLP (*Service Location Protocol*) for the location of printers and other useful info. Existing proxies (relaying and gateway−providing computers often located on the borders between Intra−Nets and the Internet) can route IPP data without change in their own source code and with only minimal re−configuration.

Operating Systems (Server, Client and Standalone):

CUPS and ESP PrintPro are available from Easy Software Products for various Linux and UNIX platforms, especially for...

- ...all Linux versions from kernel 2.0 (on Intel processors);
- ...SUN−Solaris (for Intel and SPARC processors) from version 2.5;
- ...HP−UX from version 10.20;
- ...Silicon Graphics IRIX from version 5.3;
- ...Digital Unix from version 4.0;
- ...Compaq Tru64 Unix from version 4.0;
- ...OSF/1 from version 4.0;
- ...IBM−AIX on PPC from version 4.3.

CUPS has also been ported to *BSD (though not by the CUPS developers themselves). It is also known to compile on the IBM zSeries (OS/390). ESP PrintPro− or CUPS−clients and −servers of all supported Linux− and UNIX−versions interoperate seamlessly across platforms.

Apple has announced to use a commercially licensed version of CUPS (non−GPL) in their Mac OS X (from version 10.2) and Darwin operation systems.

Why is LPD being Retired?

When LPD came into being, in the mid−70ies, the Internet as known today (with its ump−teen millions, over TCP/IP connected computers) was simply inconceivable. (Its pre−decessor did only at around 1985 achieve the number of 1.000 networked nodes). Questions of authentication and encryptions were unimportant. Printers didn't sport any "intelligence of their own": yet today's models show the capacity to compute, under the control of their own OS, with their own CPU, a color graphic in 2400 * 2400 dpi interpreting the directives of an appropriate PostScript program, save it temporarily into their internal memory and then transfer the resulting bitmap at once onto a cut−sheet medium. −− Printers in these old days were just "hammering", character by character and line by line ASCII−only (or otherwise encoded) text onto paper, which was drawn in the shape of a zig−zag folded endless paper snake from a cardbox underneath the table. Printers were not nodes in their own rights on the net, but mere attachments to a mainframe−sized computer, which occasionally let a neighboring "node" of the same size do some printouts via LPR/LPD.

LPD/LPR never was a unified standard. It was put on paper as "RFC 1179" only in 1990. In the IETF standards track it never surpassed the status of an "informational" paper. It merely served the purpose to put into writing some "best practices", which had been established by then. LPR/LPD and RFC 1179 were a minimum common demominator for already diversifying developments amongst manufacturers and software writers.

Not the First Attempt to Replace LPD

The further the printing and networking technology developed, the more deviations and vendor−specific "extensions" were put into the different LPD implementations −− and the more those implementations had problems to inter−operate. Vendors like HP even attempted to establish their own standard with "HP JetDirect". There came a stage where it was no longer bearable. In the world of UNIX and Linux various attempts had already been made to surpass the classical environment of the Line Printer Daemon: LPRng, PPR, PDQ, PLP etc. were different attempts to lift old−LPD's restrictions and put something new onto its place. But none of these attempts had a broad enough backing to succeed decisively.

The PWG initiative to overcome the burning weaknesses had more success. IPP is the result.

Change will however take a while. Too many legacy devices are still out there. Anyway, most of today's generation of network−enabled models have IPP built in already.

Backward Compatibility

Now... do you have to fear, that the broad migration to IPP will render invalid old applications, programs and shell−scripts, which rely on old−style printing? Or that you can't use legacy printers any longer?

No −− IPP is aware of the necessity to provide backward compatibility. You can safely migrate to IPP without having to throw away old applications or needing to completely re−write their printing APIs. IPP provides for a "mapping" to LPD to support the most important functions in much the same way as the old applications rely on.

CUPS for example supports both forms of print commands which are common in Linux and Unix: "*lpr*" and "*lp*" (of course these are own, separate implementations because they need to support the new CUPS−specific parameters and options too).

The CUPS Daemon

In the center of the CUPS software is the CUPS daemon *cupsd*, also called *scheduler*. It is responsible for running all relevant processes related to printing: receiving print jobs (from localhost as well as from remote computers), temporarily storing and spooling them, routing them through the correct *filters*, sorting them into the right *queues*, arranging the right order of priority, computing quotas, authenticating users, sending the jobs to their final destination by calling the right *backend*.

Of course, this is nothing new. The same sort of things are by and large done reliably by other, traditional print and spooling systems too. The three "killer" features of CUPS come from:

- the initially mentioned complete support for the Internet Printing Protocol, including the innovative web−based user interface, that allows access to configuration and print administration functions from any browser of any platform (if remote access is not denied qua configuration);
- printing from clients to all target output devices (which are configured on the server) without having to install or even configure a single driver on the client manually (anybody naming this "Zero Administration for Clients"?);
- its backward compatibility and multi−protocol−enabled support for clients, which still

rely on Windows ("Samba"), older UNIX systems ("LPD") or Apple MacOS ("NetATalk").

Drivers in the Client−Server Architecture of CUPS

In CUPS−speak every computer which has installed a "driver" and is communicating directly to a target print device is a "print server". Print servers need to have installed and use "printer drivers".

Printer drivers in Linux or UNIX can not directly be compared with those in Windows OSes. In CUPS, like in Unix traditionally, the drivers' role consists in the processing of the jobs through various "filters", which prepare the printfile format in such a way that it becomes digestible to the printer. Normally, in Unix the processing of the print jobs is often done completely automatically, with no job options (duplex, paper tray etc.) on the user side to be set. [If your system administrator is very smart, he'll have configure a few separate queues, under different names, leading to the same output device: the duplex queue will then print duplex, the a3 queue on A3 paper, etc. This workaround is very rigid and many users are not entirely happy with it...]

CUPS changes this. Here every "driver" is associated with...

- ...a printer description file (PPD, *PostScript Printer Description*), which describes in a standard way the device specific options and how to call and control these,
- ...a single filter (or a chained set of these) to convert input files into the appropriate formats that can be sent to the printer (often based on *ghostscript*, which is a PostScript interpreter in software), with the filter to−be−used named in the printer PPD,
- ...and a *backend* which is responsible for the sending of the job adhering to the printing protocol chosen for the target device (parallel, serial, USB, network/LPD, network /HP JetDirect, network/IPP, network/SMB).

In principle every CUPS computer may share his locally installed drivers on the net to other computers. These will then become the latter's "clients". The clients' jobs are *fully* processed on the server (not just spooled, but also converted and "RIP−ped" as needed).

Server−side Job Processing

A client is a computer, which normally has no drivers installed locally, but utilizes the print services (including the drivers) installed on another computer. If a CUPS box acts on the LAN as a server, it is also responsible for the complete job processing (including the filtering). CUPS therefor is geared to server−side job processing, and it is not a mere spooling device. Thus, with the help of ghostscript, it becomes a "generally available Software RIP offering its services on the LAN" (more about this further below).

Of course, you can have a client side job processing too −− but then it needs the drivers (filters and PPDs) installed locally, and the remote CUPS server is a mere spooling host. A standalone box with the CUPS printing system aboard therefor plays "its own"server, and also is a client to itself.

CUPS Servers are Web Servers on Port 631 with IPP Capabilities

The CUPS developers have intentionally chosen a similar formal structure for the configuration file of the CUPS daemon as is known from the common web server *apache*. As IPP bases itself on HTTP 1.1 this is an obvious choice. The configuration is to be found in */etc/cups/cupsd.conf*. It looks familiar to everybody acquainted with *http.conf*. There is a setting for every CUPS−specific purpose: for server name, location and names of log files, server port in use, access masks, securing server areas ("locations") or the whole issue of authentication, its methods etc. This paper will deal with some setting in more detail now.

Browsing −− Automatic Announcement of Printer Lists on the LAN

CUPS boxes keep communicating to each other over the IPP port 631 (this is conforming to the IPP standard; but you can change it via configuration). Not just print data are exchanged through the IPP−connection over HTTP, but also other info.

Here the "Browsing" plays its central role: every CUPS computer, in whose *cupsd.conf* the directive

```
Browsing On
```

is set(default), receives via UDP broadcasts the names, plus some additional infos and status bits concerning printers from the LAN in regular intervals. This info is sent out through UDP by the CUPS server(s).

If additionally the directive

```
BrowseAddress 255.255.255.255
```

(per default it is commented out now) is set, the CUPS computer starts broadcasting its own local printers to the neighborhood. A valid BrowseAddress directive turns the CUPS computer effectively into a CUPS server. (A box with the commercial CUPS version ESP PrintPro only does send broadcasts, if it has a valid license loaded.)

The broadcasting intervals are controlled by the directive

```
BrowseIntervall 30
```

f.e. to occur every half minute once. As the default BrowseAddress is 255.255.255.255, the CUPS daemon trombones his printer info across all available network interfaces into all connected LAN.

In previous CUPS configurations (like originally used in Linux Mandrake 7.2) unfortunately this setting provoked existing ISDN cards to try and dial into the Internet Provider every minute, as they tried to obey to the CUPS daemon's broadcasting directive. Linux newbies of course were often overcharged with this challenge: they didn't know how to control and limit this conduct −− but of course CUPS had a configuration option to set up the broadcasting behaviour to target exactly the intended network card or LAN: the directive

```
BrowseAddress  10.160.31.255/255.255.240.0
```

f.e. effectively limits the printer announcement to the home subnet of the author of this paper.

Volume of Browsing Info Network Traffic

To limit the mere volume of broadcast traffic, there are no more than 1024 Bytes per printer and broadcast available. In real−life environments you will normally be satisfied even with 100 to 150 Bytes (depending on how much additional descriptive info about your printer you want to provide in the announcements). Descriptions about the location of your devices and other infos are optional; the administrator might use them to comfort his users, as these strings are showing up in the GUIs or in the web interface for on the list of available devices.

Caching of Printer List on the Client Side

All CUPS daemons "listen" on port 631; they catch broadcast information from the servers and cache them locally. The lease time of their local cache is also variable; by setting *cupsd.conf* directive

```
BrowseTimeout 300
```

it will f.e. extend to 5 minutes.

Browsing only works up to the next router or gateway; according to TCP/IP specifications no broadcasts are passing across those, broadcasts are normally limited to the local LAN.

Fine Tuning the Pickup of Broadcast Info

If a client shall only evaluate and appraise the announcements of one or more particular CUPS servers (ignoring the rest), this can be achieve by lines like the following:

```
BrowseAllow 10.160.31.255/255.255.240.0
BrowseDeny 10.160.16.55
BrowseOrder Allow,Deny
```

This will specify....

- ...a range of "listened to" servers (either named, or covered by a regular expression using an asterisk as a "wildcard" place−holder)
- ...and specifically exclude particular nodes (also by naming them explicitly or covering them by a regex).

In the quoted example the client accepts all browse packages of all servers on the author's local net (which according to the used netmask spans 10.160.16−31.0−255), but specifically ignores the ones arriving from the box with the IP address 10.160.16.55. When generating the list of available printers, this one server and its printers will not appear in the list. (On the named node my fellow worker Hansjörg is occupied with all−time changing CUPS configurations and installations and sometimes dozens of different printers −− for me this would result in a way too confusing browse list.... ;−)

When name resolution is working, these directives can also be given in the form of

```
BrowseAllow .systemsupport.danka.de
BrowseDeny development.systemsupport.danka.de
```

In other words: host and domain *names* are of course also legal at the same places where you can specify IP *addresses* in cupsd.conf directives.

"Zero Administration" for Clients

Of course, from the point of view of comfort for net administrators and users the "browse" functions guarantee the easiest way to setup a well−functioning and versatile print service. Every single change on the servers is visible for all clients and binding within 30 seconds (or whatever the setting of "`BrowseIntervall...`" is):

- Your boss has approved in the budget for retrofitting the departmental printer with a a finisher and a sorter−mailbox? OK, just adapt the driver on the server to the new situation, and within 30 seconds all 200 clients can access the finishing and mailbox−sorting functions (OK, *sometimes* it may take a full 1 minute... ;−).
- The Head of Department decided that the pricey color laser printer may only be used after typing in a password? No problem −− changing the configuration on the server perpetuates to the clients in a minute....

In none of these cases any client needs to be touched directly.

A Demo turning Windows Administrators green with envy... ;−)

An amazing presentation or demo can be achieved with the following setup::

Assuming there exists a CUPS server (f.e. running on Linux) in your LAN, which has 20 different printers pre−configured. Now, on another box, like a SUN SPARC running Solaris, install (under the eyes of the spectators) CUPS, or better: ESP PrintPro. It only takes a minute or two. During the software installation you have the ethernet cable disconnected from the SUN. After the install script is finished, open the ESP PrintPro "*Printer Manager*" on the SUN desktop (root command "printers" will do it). The window (of course) is empty −− no printer there.

Next, re−plug the ethernet cable into the machine. (You can mumble "*Acra−ca−Dabra*" after having done so if you like to maximize the show ;−)

Wait for a few seconds. After re−connecting to the LAN, the SUN−CUPS client receives the broadcast infos from the server and starts to fill the window. Within seconds a list of your 20 printers builds up, seemingly from the "Nirvana". This already will catch the minds of your audience.

Now proceed to configure a pretentious job on your most advanced printer: duplexed, stapled, punched, with a front cover sheet from a different paper tray (the one holding the red paper). [But make sure the printer is switched on and not down for repair ;−)]

The "Aaahs" and "Ooohs" and general astonishment will not go away very quickly... Why, oh why is on every such occasion someone mentioning the buzzword "Zero Administration for Clients" towards the end ?

Polling −− Active Procurement of Printer Lists by the Clients

There might be cases where an administrator does not desire the printer browsing to happen. (For the sake of the broadcast traffic associated with automatic browsing, it needs to be pointed out that it is minor as compared to the noise which two windows boxen are shouting on the wire while they are fighting out their regular "master browsing elections".)

Anyway, here you may order the clients to fetch the printer infos actively themselves by "polling" the responsible server. Of course, this is now more work for the administrator to be done: every single client needs to be dealt with. The directive

```
   BrowsePoll 10.160.16.55:631
```

causes the client to inquire for available printers at the named address and port as soon as it needs to print... (To be honest, I would *never−ever* use this particular polling address, as it is a server run by my chaotic fellow worker Hansjörg.... OK, I'm stopping it now, you know it already from reading above ;−)

`BrowsePoll....`"−directives may appear multiple times in the *cupsd.conf*; the client then just polls all the named servers..

Polling is indispensable for other cases: if you need to access a server, which is at home behind a router or gateway (i.e.. in a different sub−net). See the next paragraph.

BrowseRelay −− Utilizing Servers from other LANs locally

A broadcast fails across routers or gateways. One recipe to overcome this obstacle is to make the clients poll the server(s). Now do you *really* want 500 clients polling a remote server? And configure them all to do so? And troubleshoot, if it is not working? Thought so...

You need other means to utilize printers and servers in a neighbouring LAN, whose broadcasts are not directly coming through. A simple and effective method is to appoint one (or more) CUPS computers to become "*BrowseRelays*". You can use a computer which is at home in two or more subnets to relay browsing info to the other nets as needed. For example, look at an entry of the form:

```
   BrowseRelay 192.78.215.80 10.160.31.255/255.255.240.0
```

It will record all announcements of the remote server 192.78.215.80 and its printers and pass this info as a broadcast into the other net 10.160.16.0 (which is "subnet"−ted with netmask 255.255.240.0 in my case).

Sometimes this method is not feasible (f.e.. your router might not be a Linux or CUPS box at all). No problem: you can also make any one CUPS box poll one or more servers directly and ask it to broadcast this info to its local LAN. This is an elegant alternative to have all clients poll for themselves. Look at these two entries as an example:

```
  BrowsePoll 192.78.215.80:631
  BrowseRelay 127.0.0.1 10.160.31.255/255.255.240.0
```

They suffice to have the computer poll the remote server 192.78.215 and share its own complete knowledge (localhost 127.0.0.1) about printerlists to the LAN. (Clients don't go through the BrowseRelay box if they need to *print*; they rather connect directly to the remote print server, which is now known to them from the relayed broadcasts).

"BrowseRelay"−directives may appear multiple times in the configuration file.

Interfaces to the Clients:

These interfaces serve to receive incoming printjobs and provide communication with the clients (regarding status messages etc.):

- for ESP PrintPro− or CUPS−clients (Linux− and supported UNIX platforms) the default protocol is IPP;
- Linux−, Apple− and UNIX−clients may optionally also be served via LPR/LPD;
- for older Windows−clients (Win9x, WinNT4.0) you may electively choose LPR/LPD (RFC 1179), SMB/CIFS (standard networking protocol of Microsoft systems, here implemented by Samba) or IPP (if you instate an IPP client software here);
- as Microsoft did implement the IPP 1.0 (*not* IPP 1.1!) for Windows 2000 and Windows XP in a sufficiently compatible way, these type of clients may, without further software−installation, connect to ESP PrintPro and CUPS for printing (authentication is not supported however);
- for Apple−Computers you may choose to connect via LPR/LPD, IPP or AppleTalk (in the case of AppleTalk you need to install the package NetATalk on the server side).
- newer Apple computers (using Mac OS X from version 10.2) or a recent release or Darwin are able to be native CUPS clients (for details about the setup and possible tweaks see *http://www.danka.de/apple−cups−en/*)

Backends −− Communication from Server to Printer

Printers may be attached to the server using different interfaces and protocols: parallel or serial, via USB− or network connection (over IPP, LPD, HP JetDirect or SMB/"Samba" amongst others). This task is assigned to the so−called CUPS *backends*; for every protocol a different backend is responsible (insofar backends roughly conform to what is known as "*Port Monitors*" in Windows).

Here is a list of the network backends with a brief description of their purpose..

the IPP backend:

it routes jobs to other CUPS print servers, to IPP−enabled printers or other IPP−speaking print servers. Momentarily the market knows of more than 300 devices with IPP−support built in. Amongst them are all internal and external HP JetDirect print servers of the types 300X, 400N, 500X and 600N (as are used f.e. inside the LaserJet 8000 series) as well as the complete new generations of devices from the vendors Axis, i−data, Lexmark, SEH and many more. [Many implementations are IPP−1.0 however].

the LPD backend:

it is used to talk to all legacy network printers (or other print servers, who need to receive a job from CUPS for whatever reason); LPD is so−to−speak still the "common denominator" for all network print devices.

the socket backend:

it talks fluently "HP JetDirect" (a.k.a. *AppSocket*−Protocol); therefor it is the first choice to try if the target printer is capable to use this protocol (which surprisingly many non−HP devices are!), because it sports some bi−directional abilities and enables certain status feedback messages from the printer, which are not available from LPR/LPD.

the SMB backend:

shared printers, which are attached to Windows computers may be fed with printjobs via Samba. However, you need to install *Samba* from version 2.0.6 or later on the CUPS−Server. The tool "*smbspool*", shipped as part of the Samba suite (and also written by Michael Sweet) will do the job if it is sym−linked to the *smb−backend*. Create this sym−link by typing (as root)

```
ln −s `which smbspool` /usr/lib/cups/backend/smb
```

into a terminal window.

Thus the "built−in" backends at large in CUPS allow to send jobs to any type of traditional printer or server; after all, these devices won't disappear over night just to do a favor to CUPS and IPP... ;−)

Backends:

The *backends* serve to send printfiles and overview their transfer to the output devices or to other servers on the network. They are available for different hardware interfaces and transfer protocols. Future extensions are easy to add, adapting to the development of technology:

- ***serial*** connections ("COM ports" in the Windows world);
- ***parallel*** connections;
- ***USB−***Interfaces;
- various. ***Network*** connections (covering the protocols HTTP, IPP, SMB/CIFS ["Samba"], AppSocket ["HP JetDirect"], AppleTalk ["NetATalk"])

All backends are at home in the directory */usr/lib/cups/backend/*. Amongst them are separate backends for local serial, parallel and USB interfaces as well as backends that support various protocols over an ethernet connection.

Clients from the Apple−, Microsoft− or remaining UNIX realm:

Non−CUPS clients from a UNIX−, Apple MacOS− or MS Windows−environment can also access the printing services of a CUPS− or ESP PrintPro print server. Which protocol to use for the purpose depends on the preferences of the administrator or on outside requirements:

- over AppleTalk: Apple MacOS;
- over the traditional LPR/LPD Protocol: Apple MacOS, UNIX, MS Windows and others;
- over SMB/CIFS ("Samba"): MS Windows−versions of all generations;
- over IPP: MS Windows 2000 natively, as well as all operating systems, which have any working IPP−Client−Software installed.

Potentially the installation of additional software on the client or the server is needed. This software doesn't ship as part of CUPS or ESP PrintPro (like for example the OpenSource package Samba).

PostScript requires Interpretation

Nearly all applications which are able to produce printable output in Linux (or Unix) do so by using PostScript. PostScript is a programming language, which is specially fit to describe the graphical page representations of all kinds of documents in an abstract way. PostScript uses a special syntax, which defines lines, curves, fillings, letters, fonts and many more graphical objects as well as their actual position, size, scaling, rotation and color to result in a complete page description how the printed paper should look like. PostScript therefor is very close to a "vector"−format (although it can of course also contain "ready−rendered" embedded bitmap elements).

However, the "*Marking Engine"* of a printer will only lay on paper the page by going through it "dot by dot". A black line in this realm is only a marshalling and serialization of very densely set, single pixels. As PostScript doesn't describe the arrangement of toner or ink on the paper based on their pixel positions, the adequate "raster" image format for the printer must be generated by a computation called the "*Raster Image Process*" (RIP).

Embedded in PostScript−printers are *hardware−RIPs* in the shape of specialized PostScript−processors. Therefor you may send an "un−interpreted" PostScript file (=PostScript program) to PostScript printers directly. They are capable to compute from the PostScript−files the raster images quickly and precisely and pass it on to their Marking Engine...

ghostscript −− A Software RIP for Non−PostScript Printers

For Non−PostScript−printers you must pre−digest the PostScript print file. It needs to become a file in the raster image format which fits the needed input type of the target Marking Engine *before* it is sent over. Under Linux (and many commercial Unices) this is the responsibility of the well−known *ghostscript*−package; ghostscript therefor may be described as a *software−RIP*, sitting on a place outside the printer.

As every vendor uses his own, proprietary raster formats for his printers (and keeps changing them from model to model) there is a huge multitude of available ghostscript "filters". These filter programs do the actual file format conversion.

Especially delicate is the task in relation to modern inkjet printers. Some of these contain already 7 different ink tinges (complementing the three base colors with additional lighter versions each to create more perfection in light tones, plus black). Moreover, they can spray droplets of variable sizes onto the paper to get every all−so−fine color gradation onto the medium to reach photo−like quality in printing.

The art of designing inkjet drivers (without having access to the hardware specification) lies not only in the computation of the mere 2−dimensional array of bitmap with the right amount of the required color on each pixel. The order of transfer for the computed, individual pixel−"line" data to the printhead is equally important. Modern printheads have many jets (=lines) spraying at the same time with each horizontal movement of the head. The picture is therefor built up in a process that reminds of "weaving".

So for driving the inkjets to photo−like quality doesn't just require an extremely good "dithering algorithm", but also the correct timing when the lines of pixel data are send to the printhead. The printers normally don't have much RAM built−in and therefor the printheads need the data fed to them "just in time".

Vendor specifications about raster image formats and their transfer to the printheads of their models are mostly not available to the developers of ghostscript filters. Therefor it is extremely difficult to score usable results, let alone perfection. Many modern inkjets therefor used to produce an appropriately bad result in color printing for a long time.

Excursion: The Gimp−Print−Project

This will decisively change very soon −− and it has changed already, thanks to the Gimp−Print−Project. This project has made a wonderful start already with the release of its *stp*−driver. "stp" already drives a variety of Epson, Canon− and HP−printers to spit out photo−like quality...

Gimp−Print originated around the development of a mere print plug−in for GIMP, the well−known image manipulation program. [By the way, Gimp−Print was also started by Michael Sweet, who now is the lead developer for the CUPS effort. Since working on CUPS, he passed responsibility for Gimp−Print on into the hands of Robert Krawitz; Michael however still contributes to the Gimp−Print code base.]

Gimp−Print can be downloaded from http://gimp−print.sourceforge.net/ as a source package. From this code base you may compile the stp filter in three different incarnations:

- as a ***plug−in for the GIMP*** (enabling the generation of raster code for more than 100 printers directly *inside* GIMP, from where it then can be sent in the form of "raw" print files to any spooler);
- as a ***"normal" ghostscript filter*** (which then requires as a supplement Grant Taylor's *cupsomatic* to be able to integrate stp with CUPS),
- as a ***dithering library*** providing its rasterization functions in a generic way to interested program(mer)s for usage;
- and as a ****native* CUPS printer driver***, so that it seamlessly integrates into CUPS, supplying the built−in driver functions for any other (non−GIMP) program (here it brings with it more than 100 different PPDs for HP−, Canon− and Epson inkjet printers).

Which of the 4 versions is compiled by "make" is controlled by parameters for the *configure* script. Look at this "configure":

```
./configure               \
  --with-gimp             \
    --with-ghost          \
     --enable-libgimpprint \
        --with-cups       \
```

It will cause the "`make`" command to generate all varieties of Gimp−Print at once.

pstoraster −− At the Heart of The CUPS Filtering System

The central CUPS filter is called "*pstoraster*". It does what its name suggests: convert PostScript input to a raster format. This raster format is a generic bitmap format as specified by the CUPS developers. "pstoraster" is largely based on GNU ghostscript 5.50, enhanced by the CUPS developers by taking everything out what is not needed. Work is on the way to port pstoraster over to base itself off ghostscript 6.51. Despite the seemingly old ghostscript version, CUPS is producing amazing quality on many inkjet printers −− far better than the comparable stock ghostscript version (or even the current). [*BTW, to avoid a FAQ being* *asked: you don't need to install ghostscript 5.50. CUPS brings everything it needs. And no, you don't need to de−install you recent Aladdin ghostscript−7.0.4 or GNU ghostscript 6.53 from your system, they don''t conflict with CUPS*].

CUPS can also process many more file formats other than PostScript: PDF, International Text, TIFF, GIF, PNG, JPEG, SUN−Raster and many more graphic formats may be sent directly to the spooler, and it will know how to process these. Filters with the obvious names of *pdftops*, *texttops*, *imagetops* or *imagetoraster* will be put to work accordingly.

However, CUPS needs a reliable way to identify and catalog the received files from an unknown type or source.

MIME−Types −− Determination and Assignment of File Formats

This purpose is served by the use of *MIME types*. MIME types are used for identifying file types in a non−ambiguous way (also independent from their attached file name extensions) and decide what to do and how to process the identified type of file (i.e. which filter shall deal with it).

MIME stands short for *Multipurpose Internet Mail Extensions*. MIME types are defined in a RFC standard paper. MIME types for newly developed file formats may be registered, reserved and published with IANA (*Internet Assigned Numbers Authority*).

MIMEs originally were created to standardize the exchange of files of all kinds via e−Mail (which initially only tolerated printable ASCII characters). Meanwhile MIME types also play their role to identify data which are transferred, played and saved via HTTP (surfing the web you come across all sorts of files, not just plain HTML).

When sending a file via HTTP the server tags it with a non−ambiguous MIME type. A client browser receiving this file with the correct tagging which is not able to show it up on screen by itself, "knows" from the MIME type name which program or plug−in it should call for help to open the file. (You can often change the MIME types related settings in your browser). Although there are programs out there which try to identify a file type from its name's extension, this is in no way reliable (even if you exclude deliberate mis−naming): the extension *.doc* f.e. may indicate a simple text file, a FrameMaker− or an MS Word document...

MIME types remove any ambiguity by registering them with IANA. Participating programs (browsers, web server, mail clients, mail server...) do have a unequivocal scheme to identify files and exclude misunderstandings.

MIME types are indicated according to the following scheme:

```
major_category/minor_category.
```

Examples for MIME types used by CUPS are *text/plain* (pure text file), *text/html* (HTML file), *image/gif* (graphic in GIF format), *application/pdf* (PDF file), *application/vnd.cups−raster* (the CUPS−internal raster format, which is also registered with IANA).

Filtering Based on MIME Type Assignment

CUPS uses the configuration file */etc/cups/mime.types* to define the rules, which are applied to unknown file types in order to get a MIME typization for them. When passing the job files to filters or other IPP computers these MIME types provide a reliable tagging about the file content.

Examples: the line

```
 image/gif        gif string(0,GIF87a) string(0,GIF89a)
```

describes the rule "*Any file with the strings 'GIF87a' or 'GIF89a' at its very beginning is an GIF image file*". The line

```
 image/png        png string(0,<89>PNG)
```

describes the rule "*Any file with the HEX value of '89' followed by the string 'PNG' is a PNG image file*". The line

```
 image/x-xpixmap  xpm ascii(0,1024) + string(3,"XPM")
```

describes the rule "*Any file which contains only printable characters within its first 1024 Bytes while the string 'XPM' is contained at offset 3 from the beginning of the file, shall be deemed as an image of type XPM*".

And now for the last piece of the puzzle: */etc/cups/mime.convs* is the configuration file determining, which filter is responsible for which MIME type...

Examples: the line

```
 application/postscript         application/vnd.cups-postscript 66 pstops
```

determines the *pstops* filter to deal with MIME type *application/postscript*, ask for a cost of *66* for the job, and produce MIME type *application/vnd.cups−postscript* as the outcome. The line

```
 application/vnd.cups-postscript application/vnd.cups-raster   100 pstoraster
```

determines to ask for a cost of *100* if the type *application/vnd.cups−postscript* is to be sent through *pstoraster*, and take its outcome as type *application/vnd.cups−raster*. The line

```
 image/*    application/vnd.cups-postscript 66 imagetops
```

orders all images which pass imagetops to be taxed the cost of "*66*" and produce output type *application/vnd.cups−postscript*. The line

```
 image/*    application/vnd.cups-raster    100 imagetoraster
```

asks for all image types passing *imagetoraster* for a cost of *100* to produce the outcome of *application/vnd.cups−raster* .

Every supported MIME type has a line in the *mime.types* file. Every CUPS filter has a line in the *mime.convs* file to decide about its tasks and the related costs.

Now what is the matter with these obscure "costs"?? −− If CUPS is trying to construct a working filtering chain, there are often *different* valid combinations possible. It then decides for the "route of least cost". For example, an image could go through *imagetoraster* only (at a cheap cost of 100), or through a full chain of *imagetops −−> pstops −−> pstoraster* (at a combined cost of 232). If you want your images to go this way (maybe because you want to have the more precise accounting from pstops?) just raise the price for the imagetoraster filter above the level of 232 (by writing the value into this configuration file) and you are done!

Now the picture is complete: you get the idea about how the modular design of CUPS allows for any future extensions. Just enter new MIME types and filters into the configuration files and CUPS will be able to use them. Adust the virtual "costs" assigned to the filters to give preference to one and disinclination to another...

Filters −− Workhorses of the ghostscript RIP

The collection of the various CUPS filters make up the full RIP functionality. It enables to process and print many different input files. Often you can guess from the names, which kind of conversions they conduct. On my system (CUPS in the extension of ESP PrintPro) in their home */usr/lib/cups/filter/* the following can be found:

**** hpgltops***
: Conversion of the HP plotter language HP−GL/2 to PostScript

**** imagetops***
: Conversion of various graphic and imaging formats to PostScript

**** imagetoraster***
: Conversion of various graphic and imaging formats into CUPS−inherent raster format

**** pdftops***
: Conversion of PDF (Portable Document Format) to PostScript

**** pstops***
: Conversion of PostScript to PostScript (needed to count the pages of jobs, to impose a layout 2−up or print pages selectively from a PS−file)

**** pstoraster***
: Conversion of PostScript into the CUPS−inherent raster format

**** rastertoescp***
: Conversion of the CUPS−inherent raster format to ESC/P (Epson format) [only with ESP PrintPro]

**** rastertoescp2***
: Conversion of the CUPS−inherent raster format to ESC/P2 (Epson format) [only with ESP PrintPro]

**** texttopcl***
: Conversion of Text into PCL [only with ESP PrintPro]

**** rastertopcl***
: Conversion of the CUPS−inherent raster format to PCL [only with ESP PrintPro]

**** rastertoepson***
: Conversion of the CUPS−inherent raster format to ESC (Epson format) [only with CUPS]

**** rastertohp***
: Conversion of the CUPS−inherent raster format to PCL[only with CUPS]

**** texttohp***
: Conversion of Text into PCL [only with CUPS]

For most conversion tasks various filters need to be concatenated to generate from the given input the device−specific output file which the backend then can transfer to its destination. "pstops" is most of the time involved, even if it is just to provide for the accounting functions...

The chain of concatenated filters looks like this for the simple printing of the CUPS test file (which is pure PostScript) on a HP LaserJet: *pstops −> pstoraster −−> rastertohp*. When printing a GIF graphic...

- ...the "chain" consists of one member only: *imagetops* in the case of a PostScript−printer;
- ...if printing to a LaserJet from CUPS it is *imagetoraster −−> rastertohp*,
- ...and it is *imagetoraster −−> rastertopcl* if I use ESP PrintPro.

Printable Formats:		
Many file types can be printed from CUPS in a completely "transparent" way, i.e. without needing to first open an application that is able to read and write the file type. These formats are classified as MIME types and are automatically detected and converted through filters in such a way that a printable format is generated for the target printer. Amongst the supported formats are:		
Adobe® PDF Adobe® PostScript® (Level 1 − 3) ASCII Text Graphics Interch. Format (GIFSM) HP−GL HP−GL/2	International Text JPEG/JFIF Kodak PhotoCD Portable aNyMap (PNM) Portable BitMap (PBM) Portable GrayMap (PGM)	Portable Network Graphics (PNG) Portable PixMap (PPM) Sun® Raster Tagged Image File Format (TIFF) Windows® Bitmap (BMP)

Modular Architecture: Easy for Vendor to Plug in Drivers

The modular design of the filtering functions provide for the easy plugging in of additional filters into the CUPS framework at any time. Printer and software manufacturers may very simply and for a reasonable price develop "drivers" for Linux and Unix. They only need to create a filter of their own (or adapt an existing one), write a printer PPD and possibly supply one additional entry in the file */etc/cups/mime.convs*.

Filters don't even need to be supplied in source code, if the vendor thinks he needs to protect his "intellectual property" this way. Filters will work also as binary−only plug−ins.

Only one condition must be met: CUPS filters need to...

- ...follow the standard behaviour of any UNIX filter (read from *stdin*, write to *stdout*)
- ...and be able to process 6 or 7 further parameters from the commandline: user name, job title, job−ID, printer name, number of copies and the committed file name (if it is the first filter in the chain).

One example for a vendor to supply his drivers as a CUPS plugin is TurboPrint. It is a shareware program, initially developed for the Amiga platform, and able to produce photo−matching quality on many current inkjet printers. It also contains some basic features of color management.

Dedicated Print Servers

If CUPS is to act as a dedicated print server responsible for an amount of clients, you should make sure the system is fit for the processing of the expected print volume. Handling PostScript by the *ghostscript software RIP* requires possibly (depending on the target printer) a lot of resources: CPU time, RAM, HDD space. Raster data for an A3 full color page on a 6−color inkjet may easily surpass 200 MB of data.

Because in the case of CUPS, where the clients are relieved from the task *Raster Image Processing* and the server provides this for them (as well as relieving the administrator from a lot of configuration work on the clients), it should never be too weak for the projected print volume. Otherwise undesired periods of waiting for the jobs to finish might occur.

If you are running PostScript printers, of course the biggest part of the CPU−intensive RIP processes are avoided on the server, as they are done by the built−in RIP on the printers. Here you'll possibly be OK with a shelved Pentium I to spool and temporarily store the printfiles.

Important Hints for First−Time−Installations

Installing CUPS for the first time on a system, where the OS vendor doesn't provide ready−made packages: I suggest to use the "portable" *.tar.gz*−format installation packages which were prepared by the CUPS developers themselves (there are also RedHat−*.rpm* − and Debian−*.deb*−packages offered on the web somewhere). Make sure to read the README and INSTALL files before you actually install CUPS..

You should remove the old printing software beforehand. (On a SuSE Linux this would be "*lprold*" or "*lprng*"). After expanding and extracting the tarball by using

```
   tar −xvzf cups−1.1.6.tar.gz
```

(this may vary on other Unix systems) you'll find in the directory the installations script. It has to be called by root using

```
   ./cups.install
```

Attention: The installation replaces existing files and commands like *printcap*, *lp*, *lpr*, *lpq*, *lpc*, *lpadmin* by its own versions, but tries to save the existing ones under the file name extension "*.O*" (Should you ever try to get rid of CUPS again, the de−installation script "`cups.remove`" will try to restore your original configuration).

Important Hints for Upgrading

Those who always want to live on the "bleeding edge" will either regularly compile from the sources themselves or fetch the binaries from the CUPS−Website packaged as a *.tar.gz*−tarball. Installation with the help of the included *cups.install*−Shellscript is a "snap". It leaves all existing configuration files normally alone and doesn't attempt to overwrite them. A new configuration file shipping with the download will be saved side−to−side to the "active" configuration file and carry the name *cupsd.conf.N* .

Attention: This means that you will have a running system, but you might not enjoy any newly released features. Therefor have a close look into *cupsd.conf.N* and transfer and adapt "manually" the possibly discovered new configuration directives into your active *cupsd.conf*... CUPS is developing very fast and in nearly each new release you will find new features and configuration options.

Caution: There is no guarantee for the retaining of the described behaviour of the installation script. It is therefor in any case advisable to make a backup of all the configuration files in */etc/cups/* (mainly the *printers.conf* and the *cupsd.conf*).

Three Different User Interfaces

CUPS disposes two different "builtin" user interfaces: the *command line,* and the *web interface* accessible from any browser.

The commercial CUPS version, ESP PrintPro, includes a GUI (*Graphical User Interface*). In addition it includes and bundles more than 2.500 printer drivers and also provides some enhancements to the ghostscript components of CUPS (these improvements are drawn from info acquired through NDA, *Non Disclosure Agreements* with various printer manufacturers). ESP PrintPro' s GUI runs on all supported platforms (Linux and Unix) and doesn't need any other toolkit installed apart from an X Window system.

Additionally, there have been other optional GUI frontends to CUPS around for 2 years now, often contributed from third party developers. *XPP* (X Print Panel) by Till Kamppeter, *QtCUPS* and *KUPS* (by Jean−EricCuendet and Michael Goffioul) and *GTKlp* (by Tobias Müller) are perhaps the best known examples. If you use the program name of one ofthese tools inside other programs as "print commands", then for every printjob the frontend pops open and the user can choose his printer and settings in a flexible and comfortable way.

KDE−3.0 with KDEPrint as a Full−Featured CUPS Frontend

Michael Goffioul from KDE has now gone one step beyond this: he has deprecated QtCUPS and KUPS and now designed and released (with KDE 3.0) a complete new framework for printing. KDEPrint−3.0 is flexible enough to base itself on different print engines. If used as a frontend to the CUPS print engine, KDEPrint allows you to construct and run an enterprise level printing system with full graphical print job creation, job option setting, printer management, job administration and control, including such advanced possibilities to have job accounting, printer quotas and more.

Various CUPS Resources and Locations

Just like a "normal" web server the CUPS daemon rules over different (real and virtual) resources (also known as *locations*), which are addressed from the clients via a URI. Amongst them are:

- /admin/,
- /printers/,
- /printers/printername,
- /printers/printername.ppd,

- /jobs/,
- /classes/,
- /classes/classname

Locations may be "entered" via any browser, if you extend the URI path from the "root" of the webserver by the location's denotation.

Examples: the location */classes/all_laserjets* on the CUPS server *transmeta.danka.de* is entered via the URL

```
http://transmeta.danka.de:631/classes/all_laserjets.
```

The PPD for the printer *DigiMaster9110* will be shown by going to

```
http://transmeta.danka.de:631/printers/DigiMaster9110.ppd.
```

Differentiating Access Rights

Of course not every location is necessarily accessible in the same way as the rest. Access rights may be different for each location. The credentials for each location will be checked prior to allow admittance. The IPP server CUPS provides -- as any HTTP server -- to the administrator the option to change the access rights for each location as needed.

The "AuthType"-Directives are for the configuration: "`AuthType None`" waives any Authentication., "`AuthType Basic`" uses the HTTP basic method and "`AuthType Digest`" uses HTTP digest

On top, access may be denied or allowed according to the descending host of the seeker. Directives similar to the ones known from the apache configuration, like "`AllowFrom...`" and " `DenyFrom...`", including the order of their evaluation "`Order Allow,Deny`" resp. "`Order Deny,Allow`" are working in much the same way here. Variables for these directives may be *All*, *None*, host name, IP address and domain name including the wildcard "`*`".

All these settings are easily controlled by editing the well commented */etc/cups/cupsd.conf*.

After a fresh installation the location */admin/* is only accessible by root, but not remotely, only from localhost (plus, the root account may not have an *empty* password!).

Attention: don't fool yourself into thinking these settings were only valid for the web interface! No, they are valid for *any* kind of access. Even if you use the command line or a GUI, you are necessarily accessing one or the other *location* of the CUPS server and you are therefor bound to what the access right for these locations are like.

Encryption over SSL3/TLS

One of the newer features of CUPS is its support for the encryption method TLS (*Transport Layer Security*). TLS has achieved now the IETF status of a "proposed standard". TLS is derived from Netscape's SSL3 (*Secure Socket Layer Version 3*). It implements an asymmetrical encryption of data. CUPS bases itself for the core functions on this sector on the developments of the *OpenSSL* project, whose libraries it uses. The CUPS developers dub their SSL support still as "experimental". If the pace of development keeps up, you can expect the moving of this label to "fully supported" soon.

Controlling Job Options From the Commandline

To use any printer−specific option from the command line, you need first to know: how did the vendor code this option into the PPD?

A good tool for this is the command "*lpoptions*". Look at the next example. Use the command on one single line, with the backslashes "\" removed:

```
lpoptions            \
  -h transmeta       \
    -p DANKA_IS2100  \
      -l
```

It will ask the CUPS host *transmeta* for the PPD options of the printer *DANKA_IS2100*. The switch *−l* (for "*long output format*") will make sure to retrieve the full set of options in a "verbose" mode. The full output for this example is too long to be quoted here. Therefor I try to search for one option only: Duplex printing. How is this printer told through a PPD option if it shall print duplex? (and: is it ***D***uplex or ***d***uplex?):

```
lpoptions            \
  -h transmeta       \
    -p DANKA_IS2100  \
      -l | grep uplex
```

produces this output:

```
RI−Duplex/Duplex: *DuplexNoTumble DuplexTumble None
```

This tells me:

- the name of the option in question here is "*RI−Duplex*" (aa−ha! it is not just simply "duplex"... yes, vendors *do* have their liberties);
- after the "slash" follows the translation of the option, as it should show up in the web interface or any GUI ("*Duplex*");
- the option may assume the three values of "*DuplexNoTumble*" or "*DuplexTumble*" or "*None*";
- the presently assumed value (the current default for the option) is "*DuplexNoTumble*" (recognizable from the tagging with the *asterisk/star* "*").

NOTE: the lpoptions command works for remote servers too. However, you need the appropriate access rights to retrieve the info (i.e. you must be allowed to print to the printer in question).

If I wanted to print simplex (deviating from the default) to the printer, I would need to use the following command:

```
lpr                      \
  -P DANKA_IS2100        \
    -o RI−Duplex=None    \
       /path/to/printjob
```

There is a third party utility which produces an output with the same content as the *lpoptions* command, but much more nicely formatted. It is called *lphelp* and was written by Till Kamppeter from MandrakeSoft:

```
lphelp DANKA_IS2100

 [....]
 Duplex:  -o RI-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:

    w774h1116  (8K, size: 10.75x15.50in)
    [....]
```

Accounting -- The Benefits

CUPS "keeps its books" about every single printed page. For each printed page there occurs an entry in the file */var/log/cups/page_log* in the following form:

```
printername   user  job-ID    [date+time]    page-number   copies  billing
```

To quote a recent example from my own page_log:

```
DANKA_LaserJet_8100 kurt 3084 [02/Jan/2002:06:23:37 +0200] 1 15 #marketing
DANKA_LaserJet_8100 kurt 3084 [02/Jan/2002:06:23:37 +0200] 2 15 #marketing
DANKA_LaserJet_8100 kurt 3084 [02/Jan/2002:06:23:37 +0200] 3 15 #marketing
DANKA_LaserJet_8100 kurt 3084 [02/Jan/2002:06:23:37 +0200] 4 15 #marketing
```

This example is *printjob-ID 3084*, printed by *user kurt* on *DANKA_LaserJet_8100*; it had *4 pages* with each printed *15 times* and its billing tag is associated with the account "*#marketing*".

Page log files may easily be processed as needed for accounting and billing purposes. There are a few utilities available already:

Thies Moeller has written the Perl script *PrintAnalyzer*. It extracts some statistical analysis and tables from the *page_log*. It tells you the figures about the number of jobs (per printer and total), the users and their respective print volume, the distribution of the print load over time, and much more.

A module to analyze CUPS page_log files is availabel from the general log file analyzer project *Lire*.

It should be relatively easy for people with some appropriate skills to modify the *webalizer* utility to provide a similar output for CUPS log files as it does for apache web logs.

Accounting −− The Limitations

The number of pages inside a job is detected while passing the filter *pstops*. This requires a PostScript input file which is *DSC−conformant* (DSC = *Document Structuring Conventions*). Normally this precondition is met and the count is correct. However, there are two restrictions with this approach to accounting:

First:

pstops *counts the RIP−ped pages inside the filter, not the actual sheets* leaving the final output device. So, if later the job gets lost in the printer, CUPS currently has no means to detect this. The page count then still stands as indicated by pstops.

Second*:*

some jobs are by−passing this important pstops filter, for example all Samba print jobs originating from Windows clients which use the original Windows printer driver. This driver generates print−ready data and therefor *requires* to be treated as a "raw" job by the CUPS server. If the CUPS server uses the option "`−o raw`" for printing, no filtering is occurring on its side (hence no pstops filter can count pages) and this job is counted with a default of "1" page only.

The current CUPS development cycle, leading to the release of versions 1.2.x, will improve on the precision of accounting info a lot. If possible, this information will be generated by the *backends*, using b*ackchannel data* about every finished page, retrieved from the printers themselves while printing the job. Of course, the printer needs to support this feature too. The info then will be more reliable, as it is counting the sheets actually leaving the printer, not the number of pages having been processed by the RIP. Only as a fallback (if the printer doesn't support the backchannel communication) will the old method be used.

More Infos about Important Commands

To find out more about CUPS commands, just type "`lp [Tab] [Tab]`" and see all commands starting with "*lp...*" appear (this is assuming you are using the *bash*−Shell). Do this as normal user, and as root −− both have sets of commands that are not entirely congruent. Most shown commands will be owned by CUPS. Read the *man pages* concerning these commands; the man pages might point you to a few additional commands (like *cancel* and *enable*) not starting with the "*lp...*" letters.

When your CUPS daemon is up and running, you will also find via

`http://localhost:631/documentation.html`

the complete and current CUPS documentation, both in HTML− and PDF−formatted versions. (from a remote client you need to use the URL

`http://CUPS−Server:631/documentation.html`

Should the CUPS daemon exceptionally suffer from problems, you'll probably find the files in your local file system in the path */usr/share/doc/cups/* .

Tips and Tricks

Experienced CUPS users are quickly assembling a lot of tips and tricks about CUPS usage. For example, you can use nearly all "official" IPP job and printer attributes, as defined in the IPP−RFCs, to pass them on the commandline to the CUPS daemon. Many of these parameters are not yet available through the web interface. (KDEPrint now has support for the most important ones through its GUI interface).

Those who are curious enough should read *RFC 2910*. This RFC defines specifications for *IPP 1.1 Model and Semantics*. They'll find many functions in there which work in the CUPS command line, but are not yet described in the "official" CUPS documents.

One example is the "*job−hold−until*" attribute. Try a command like this:

```
lpr                              \
  -P HeidelbergDigimaster        \
    -o job-hold-until=indefinite \
       /usr/share/doc/sam.pdf    \
```

This sends the *CUPS Software Administrator Manual* to be printed by the device *HeidelbergDigimaster9110* −− where the job shall not be printed immediately, but put on "*hold*" for an indefinite time. (The job may later be released manually or deleted befor print). "*job−hold−until*" is one of the "*Job Template Attributes*"as defined in IPP−1.1; it accepts as valid parameter values "*indefinite*", "*day−time*", "*evening*", "*night*", "*weekend*", "*second−shift*" and "*third−shift*" or the exact specification of a time in the format "*HH:MM*".

More items in the category "Tips + Tricks" have to be reserved for the "*CUPS Printing HOWTO*", or found out by yourself... ;−).

Perspectives

One of the areas the CUPS developers work on is to improve the service of CUPS in relation to Windows clients. Finally, if the developers don't boggle from the cruelties and wobblines of programming for the MS platform, they might even release a *CUPS−IPP−client for Windows*, with all the benefits of automatic printer discovery etc. which are now available for Unix/Linux/MacOS clients. It will work without the need to install Samba on the CUPS server, configure it and keep it running.

Samba support will, of course, continue too to be improved. The lead developer for CUPS, Michael Sweet, is closely co−operating with the Samba team to achieve a seamless integration of CUPS with Samba and the Windows world of clients. The first results of this can be seen in the latest Samba releases 2.2.x.

The new "*Point+Print*" technology inside Samba allows Windows users to get their driver downloaded and installed during first connection to the shared printer. With CUPS you can use this feature to make the Windows clients use exactly the same PPD as native CUPS clients and servers do. The driver download will then require a generic Adobe PostScript driver for Windows. This driver needs to be deposited for the clients' convenience in a special Samba share named `[print$]`. Details are described in Samba documents and in the CUPS man page for the "*cupsaddsmb*" command. This command (used by root) will arrange for any named printer to be shared out to Windows clients, including the preparation for PPD download to new clients.

The benefits are obvious: you'll have any Windows client to send PostScript to the CUPS server (even if the target printer is not a PostScript device!). From the CUPS−PPD, used by the client, the job options are taken. At the same time, this PPD defines the filter to be used by the server who has to process the PostScript to fit the target device. Thus you use CUPS as a *networked PostScript RIP for Windows clients* with the benefit of central job accounting (as every Windows job will pass the pstops filter too now).

Dedicated CUPS Print Servers to Relieve Windows Terminal Servers (WTS)?

Administrators of big installations with *Windows Terminal Servers* (WTS) possibly discover another beneficial perspective in this setup: the wild huddle and tangle with drivers on their server could come to an end.

[Some background: WTS often serve applications to a few dozen Windows Terminals. These terminals look to the local user just like any Windows machine. Normally they don't notice that the program they are using is running on the remote WTS and that their local machine is only serving to provide the "window to the server". If they print to their "local" printer beside their desk, in reality the printer driver is running on the WTS. A WTS serving 50 clients needs to have a printer driver for every client with a printer too.]

As is well known in these circles, a big part of the famous *Blue Screens* are originating from problems with printer drivers. It is no good for Wndows Terminal Servers to have installed too many different printer drivers at the same time. At some places statistics suggest more than 50% of BlueScreens and re−boots to be caused by printing problems. Remember, every reboot of a WTS makes several dozen people unable to work with their computers! Especially notorious for their bad effects are (despite of their official "certification") the PCL drivers from one here un−named vendor. PostScript drivers don't seem to cause any similar problem. Using one, and only one Adobe PostScript driver on a WTS to generate all jobs and send them to a dedicated CUPS server could greatly benefit the stability of the whole environment. [*NOTE: If you are interested in setting up a field test with a small or large WTS installation, please contact the author of this paper.*]

The Bottom Line

CUPS doesn't just simplify printing for the newbie and home user. CUPS is a godsend to overstretched administrators in the enterprise also. CUPS, this Jack−of−all−Trades, is at home in heterogeneous network worlds since its nascency. And here CUPS can help to advance the further conquest of the enterprises and their desktops by Linux: the nicest KDE including the most perfect K−Office will be to no avail if printing from Linux remains the kludge it still is at many places.... With CUPS this mess comes to an end.

.

.

.

.

.

.

The Author:

Kurt Pfeifle works in Stuttgart as a System Specialist. His employer is Danka Deutschland GmbH. Danka is one of the largest manufacturer−independent providers for sales and service of system solutions in the digital printing market, integrating hardware as well as software components, covering heterogeneous network environments. His job includes giving consulting and training to customers in network printing issues, including IPP, CUPS and Linux/Unix printing. He may be contacted via kpfeifle.at.danka.de.

Links:

http://www.danka.de/printpro/faq.html
CUPS−FAQ by Kurt Pfeifle, more than 150 Questions (and presently a some less Answers ;−)

http://www.danka.de/apple−cups−en/
"The Unofficial CUPS−on−Apple−Max−OS−X−FAQ" by Kurt Pfeifle

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 info about CUPS

http://www.easysw.com/software.html
ESP PrintPro download

http://www.easysw.com/documentation.html
First hand info 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

Annotation: This paper was prepared for SANE 2002, the System Administration and Networking Conference held in Maastricht, NL, 27th −31st of May, 2002. It is based on translating, editing and updating another paper that was presented at LinuxTag 2001 in Stuttgart.

Kurt Pfeifle