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

(III.) Some Theoretical Background:

CUPS, PPDs, PostScript® and GhostScript

Table of Contents

(III.) Theoretical Background: CUPS, PPDs, PostScript &GhostScript

(III.) Some Theoretical Background: CUPS, PPDs, PostScript® and GhostScript

This chapter aims to give a bit of theoretical background to printing in general, and to CUPS especially.

Basics About Printing

Printing is one of the more complicated chapters in IT technology.

Earlier on in history, every developer of a program that was capable of spitting out printable output had to write his own printer drivers too. That was quite complicated, because different programs have different file formats. Even programs with the same usage, for example, word processors, often do not understand each others formats. So, there was no common interface to all printers, hence the programmers often supported only a few selected models.

A new device appearing on the market required the program authors to write a new driver if they wanted their program to support it. Also for manufacturers, it was impossible to make sure their device was supported by any program known to the world (although, there were far fewer than today.)

Having to support ten application programs and a dozen printers, meant a system administrator had to deal with 120 drivers. So the development of unified interfaces between programs and printers became an urgent need.

The appearance of Page Description Languages, describing the graphical representation of ink and toner on sheets of paper (or other output devices, like monitors, photo typesetters, etc.) in a common way was a move that found a big gap.

One such development was PostScript® by Adobe. It meant that an application programmer could concentrate on making his program give out a PostScript® language description of his printable page, while printing device developers could focus on making their devices PostScript® literate.

Of course, there came, over time, the development of other description methods. The most important competitors to PostScript® were PCL (Print Control Language, from Hewlett−Packard®), ESC/P (from Epson) and GDI (Graphical Device Interface from Microsoft®).

The appearance of these page description languages eased life, and facilitated further development for everybody. Yet the fact that there still remained different, incompatible, and competing page description languages keeps life for users, administrators, developers and manufacturers difficult enough.

PostScript® in memory – Bitmaps on Paper

PostScript® is most heavily used in professional printing environments such as PrePress and printing service industries. In the domains of UNIX® and Linux®, PostScript® is the pre−dominant standard as a PDL. Here, nearly every program generates a PostScript® representation of it's pages once you push the Print button. Let us look at a simple example of (hand−made) PostScript® code. The following listing describes two simple drawings:

Example 1.0. PostScript® Code, handcrafted

Listing: A Snippet of PostScript "Code"

```
1    %!PS             % First 2 characters need to be '%!' (magic numbers).
2    % two boxes      % '%' introduces comments. The virtual PS-pen is asked
3    100 100 moveto   % to move to coordinate (100,100), then draw a
4    0 50 rlineto     % relative line 0 units to the right and 50 to the top,
5    50 0 rlineto     % go on with 50 to the right (0 to the top
6    0 -50 rlineto    % now 50 units straight down,
7    closepath        % close the "path",
8    .7 setgray       % switch to 70% gray value for the color to use and
9    fill             % fill the box with this color..
10   %                % First box is finished; next figure
11   160 100 moveto   % is constructed in an analogous way.,
12   0 60 rlineto     % but this time not just with horizontal
13   45 10 rlineto    % and vertical lines, but also with lopsided ones..
14   0 -40 rlineto    % (yes, 20% in PostScript stands for a more
15   closepath        % dark value than 70%).
16   .2 setgray       %
17   fill             % The closing command "showpage" tells
18   showpage         % the printer to eject the page...
```

This tells the imaginary PostScript® pen to draw a path of a certain shape, and then fill it with different shades of gray. The first part translates into more comprehensive English as Go to

coordinate (100,100), draw a line with length 50 upward; then one from there to the right, then down again, and finally close this part.Now take a paint of 70% gray, and use it to fill the drawn shape.

Example 1.1. PostScript® Code, less readable

Listing: A Snippet of PostScript "Code", as written by many PostScript−generating programs...

```
%!PS
100 100 moveto 0 50 rlineto 50 0 rlineto  0 -50 rlineto  closepath
.7 setgray fill 160 100 moveto 0 60 rlineto 45 10 rlineto  0 -40 rlineto
closepath .2 setgray fill
showpage
```

This is the same PostScript code, but written in a much less readable way. This is how often PostScript drivers or other PostSript−generating programs would write it. It still is completely "legal" code....

Beneath is the picture which would be drawn by "Ghostview" on screen or printed by a printer on paper after its PostScript interpreter had rendered it into a raster image:

Example 1.2. Rendered PostScript®

Picture: A Snippet of a PostScript "Image"

Of course, PostScript® can be much more complicated than this simplistic example. It is a fully fledged programming language with many different operators and functions. You may even write PostScript® programs to compute the value of Pi, format a harddisk or write to a file. The main value and strength of PostScript® however lays in the field to describe the layout of graphical objects on a page: it also can scale, mirror, translate, transform, rotate and distort everything you can imagine on a piece of paper −− such as letters in different font representations, figures, shapes, shades, colors, lines, dots, raster...

A PostScript® file is a representation of one or more to−be−printed pages in a relatively abstract way. Ideally, it is meant to describe the pages in an device−independent way. PostScript® is not directly visible'; it only lives on the hard disks and in RAM memory as a coded representation of future printouts.

Raster Images on Paper Sheets

What you see on a piece of paper is nearly always a raster image. Even if your brain suggests to you that your eyes see a line: take a good magnifying glass and you will discover lots of small dots... (One example to the contrary are sheets that have been drawn by pen plotters). And that is the only thing what the marking engines of todays printers can put on paper: simple dots of

different colors, size, resolution to make up a complete page image composed of different bitmap patterns.

Different printers need the raster image prepared in different ways. Thinking about an inkjet device: depending on its resolution, the number of used inks (the very good ones need different 7 inks, while a cheaper one might have use 3), the number of available jets (some print heads have more than 100!) spitting out ink simultaneously, the dithering algorithm used, and many other things, the final raster format and transfer order to the marking engine is heavily dependent on the exact model used.

Back in the early life of the Line Printer Daemon, printers were machines that hammered rows of ASCII text mechanically onto long media, folded as a zig−zag paper snake, drawn from cardboard boxes beneath the table... What a difference from today!

RIP: From PostScript® to Raster

Before the final raster images are put on paper cut−sheets, they have to be calculated somehow out of their abstract PostScript® representation. This is a very computing−intensive process. It is called the Raster Imaging Process, more commonly RIP).

With PostScript® printers the RIP−ping is taken care of by the device itself. You just send to it the PostScript® file. The Raster Imaging Processor (also called the RIP) inside the printer is responsible (and specialized) to fullfill quite well this task of interpreting the PostScript®−page descriptions and put the raster image on paper.

Smaller PostScript® devices have a hardware−RIP built in; it is cast in silicon, on a special chip. Big professional printers often have their RIP implemented as a software−RIP inside a dedicated fast UNIX® run computer, often a Sun SPARC Solaris or a SGI IRIX® machine.

Ghostscript as a Software RIP

But what happens, if you are not lucky enough to have a PostScript® printer available?

You need to do the RIP−ing before you send the print data the marking engine. You need to digest the PostScript® generated by your application on the host machine (the print client) itself. You need to know how the exact raster format of the target printers' marking engine must be composed.

In other words, as you can't rely on the printer to understand and interpret the PostScript® itself, the issue becomes quite a bit more complicated. You need software that tries to solve for you the issues involved.

This is exactly what the omnipresent Ghostscript package is doing for many Linux®, *BSD and other UNIX® boxes that need to print to non−PostScript® printers: Ghostscript is a PostScript® interpreter, a software RIP capable to run a lot of different devices.

Drivers and Filters in General

To produce rasterized bitmaps from PostScript® input, the concept of filters is used by Ghostscript. There are many different filters in Ghostscript, some of them specialized for a certain model of printer. Ghostscript filterspecialized in devices have often been developed without the consent or support of the manufacturer concerned. Without access to the specifications and documentation, it was a very painstaking process to reverse engineer protocols and data formats.

Not all Ghostscript filters work evenly well for their printers. Yet, some of the newer ones, like the stp Filter of the Gimp Print project, produce excellent results leading to photographic quality on a par or even superior to their Microsoft® Windows® driver counterparts.

PostScript® is what most application programs produce for printing in UNIX® and Linux®. Filters are the true workhorses of any printing system there. Essentially they produce the right bitmaps from any PostScript® input for non−PostScript® target engines.

Drivers and Filters and Backends in CUPS

CUPS up to 1.1.14 used its own filters, though the filtering system was based on Ghostscript. Namely the pstoraster was directly derived from Ghostscript code. CUPS had re−organized and streamlined the whole mechanics of this legacy code and organized it into a few clear and distinct modules. The CUPS pstoraster filter was working out−of the−box. It was working independently from any Ghostscript which might or might not have been installed on the system.

From CUPS 1.1.15 this has changed. CUPS now relies on a Ghostscript version, that needs a "cups" device (−sDEVICE=cups"). ESP Ghostscript 7.05 provides this. GNU Ghostscript needs a special patch to the source code and a re−compilation to work for CUPS. ESP Ghostscrpt therefor is the preferred distribution. In its version 7.05.4 it contains exactly 300 supported "devices" (compare this to stock GNU Ghostscript with 181 devices!).

Spoolers and Printing Daemons

Besides the heavy part of the filtering task to generate a print−ready bitmap, any printing software needs to use a SPOOLing mechanism: this is to line up different jobs from different users for different printers and different filters and send them accordingly to the destinations. The printing daemon takes care of all this.

This daemon is keeping the house in order: it is also responsible for the job control: users should be allowed to cancel, stop, restart etc. their jobs (but not other peoples's jobs) and so on.

Conclusion: How CUPS uses the power of PPDs

Now that you know how a PostScript® language file (which describes the page layout in a largely device independent way) is traveling to become transformed into a Raster Image, you might ask: Well, there are different kinds of raster output devices: first they differ in their resolution; then

there are the different paper sizes; it goes on with many finishing options (duplex prints, pamphlets, punched and stapled output with different sheets of colored paper being drawn from different trays, etc.). How does this fit into our model of device−independent PostScript®?

The answer comes with so called PostScript® Printer Description (PPD files. A PPD describes all the device dependent features which can be utilized by a certain printer model. It also contains the coded commands that must be used to call certain features of the device. But PPDs are no closed book, they are simple ASCII text files.

PPDs were invented by Adobe to make it easy for manufacturers to implement their own features into PostScript® printers, and at the same time retain a standard way of doing so. PPDs are well documented and described by Adobe. Their specification is a de−facto open standard.

Device Dependent Print Options

Remember, advanced PostScript® printing originally was developed for use on Microsoft® Windows® and Apple Mac® systems only. For a long time all the feature rich printing on modern devices was just unavailable for Linux® and UNIX®. CUPS changes this decisively. CUPS is very intimated with PPDs, and therefor existing PPDs can be utilized to the full by all systems powered by CUPS.

Via PPDs, printer manufacturers were able to insert device−specific hardware features into their products, for things such as duplexing, stapling, punching, finishing etc.. The printer drivers load this PPD just like an additional configuration file. Thus the printer driver learns about the available device options and how to call them; the driver also shows them in a GUI to the user. Through this mechanism you still are able to print device−independent PostScript® page description language files and specify device−dependent finishing options on top, which are added to the application−produced PostScript®.

Where to get the PPDs for PostScript® Printers

PPDs originally were not used routinely in UNIX® and Linux® systems. The vendors providing those PPDs never intended them for other than the originally supported operating systems, Microsoft® Windows® and Mac® operating system. Through it's brilliant move to fully support and utilize the existing PPD specification, CUPS now gives the power to use all features of modern printers to users of Linux® and Linux®−like systems. KDEPrint makes it's usage even more comfortable than the CUPS developers ever dreamt of.

CUPS can use original Windows® PPDs, distributed by the vendors in the case of PostScript® printers. Those normally don't cost any money, and they can be grabbed from any Windows® computer with an installed PostScript® driver for the model concerned, or from the disks provided with the printer. There are also several places on the web to download them.

Why Specially Crafted PPDs are Now Useful Even For Non−PostScript® Printers

Now you know how PostScript®−Printers can use PPDs. But what about non−PostScript® printers? CUPS has done a very good trick: by using the same format and data structure as the

PostScript® Printer Descriptions (PPDs) in the PostScript® world, it can describe the available print job options for non–PostScript® printers just the same. For its own special purposes CUPS just added a few special options (namely the line which defines the filter to be used for further processing of the PostScript® file).

So, the developers could use the same software engine to parse the Printer Description Files for available options for all sorts of printers. Of course the CUPS developers could not rely on the non–PostScript® hardware manufacturers to suddenly develop PPDs. They had to do the difficult start themselves and write them from scratch. More than 1000 of these are available through the commercial version of CUPS, called ESP PrintPro.

Meanwhile there are a lot of CUPS–specific PPDs available. Even now those are in most cases not originating from the printer manufacturers, but from Free software developers. The CUPS folks proofed it, and others followed suit: where Linux® and UNIX® printing one or two years ago still was a kludge, it is now able to support a big range of printers, including 7–color inkjets capable of pushing them to Photo Quality output.

Different Ways to get PPDs for non–PostScript® Printers

You can get PPDs to be used with CUPS and non–PostScript® printers from different areas in the Web:

- first, there is the repository at www.linuxprinting.org, which lets you generate a CUPS–O–Matic–PPD online for any printer that had been supported by traditional Ghostscript printing already. This helps you to switch over to CUPS with little effort, if you wish so. If your printer was doing well with the traditional way of Ghostscript printing, take CUPS–O–Matic to plug your driver into th e CUPS system and you'll have the best of both worlds.
- second, there are CUPS–PPDs for the more than 120 printer models, which are driven by the new universal stp driver. stp (stood originally for Stylus Photo) is now developed by the gimp–print project; it was started by Mike Sweet, the leading CUPS developer and is now available through gimp–print.sourceforge.net. This driver prints real Photo quality on many modern inkjets and can be configured to make 120 CUPS–PPDs along its own compilation. HP® Laser– and DeskJet, Epson® Stylus and Photo Color models as well as some Canon® and Lexmark® are covered.
- third, there is the commercial extension to CUPS from the CUPS developers themselves: it is called ESP PrintPro and comes with more than 2.300 printer drivers. There are even improved imagetoraster and pstoraster filters included.

CUPS makes it really easy for manufacturers to start supporting Linux® and UNIX® printing for their models at reasonably low cost. The modular framework of CUPS facilitates to plug in any filter (=driver) with minimal effort and to access and utilize the whole printing framework that CUPS is creating.

Read more about the CUPS features in the available CUPS documentation at http://www.cups.org/documentation.html and http://www.danka.de/printpro/faq.html. Also at http://www.linuxprinting.org/ is a universal repository for all issues related to Linux® and UNIX® printing.

About the Author:

Kurt Pfeifle works in Stuttgart as a System Specialist for Danka Deutschland GmbH, 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. 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 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

Annotation: this paper is based on a chapter in the "KDEPrint Handbook" available at http://printing.kde.org/.

Kurt Pfeifle