Free Conference on the LinuxTag 2003 in Karlsruhe, Germany

linuxprinting.org and Foomatic

The New Generation of Printing with Free Software

Table of Contents

linuxprinting.org and Foomatic – The New Generation of Printing with Free Software.....1
Why Foomatic?.....1
How did linuxprinting.org and Foomatic emerge?.....2
CUPS makes its appearance.....2
How CUPS uses the original PostScript PPD files.....2
How CUPS extends use of PPDs to the non–PostScript printer world.....3
Not enough supported printers.....3
CUPS–O–Matic and Foomatic –– the first incarnations.....3
PDQ–O–Matic next.....4
Creation of "Linuxprinting.org".....4
MandrakeSoft goes CUPS!.....4
The Database grows up.....4
From Postgres to XML.....4
C instead of Perl: huge speed improvements for Foomatic.....5
Gathering more support for Foomatic and the Database.....5
The Foomatic system as it is now.....6
The XML database.....6
How does the database work? – A small example.....6
What is done to set up a print queue with this data?.....8
How does printing with Foomatic work?.....9
The structure of the XML database.....10
An option entry: "PageSize" (db/source/opt/2.xml).....10
A printer entry: HP LaserJet 4000 (db/source/printer/HP–LaserJet_4000.xml)...15
A driver entry: "md2k" (db/source/driver/md2k.xml).....19
Composite Options.....20
Forced Composite Options.....24
String and Password Options.....25
Option Grouping.....29
Unprintable margins.....31
Adding arbitrary extra entries to the PPD file.....32
What is planned for the future.....33
Web interface for adding new printer entries on linuxprinting.org.....33
Printer/Driver classes.....34
Option conflicts.....34

linuxprinting.org and Foomatic – The New Generation of Printing with Free Software

Till Kamppeter

Why Foomatic?

Formerly, all GNU/Linux distributions and other Unix–style operating systems used LPD as the printer spooler. This is technology of the 70th made for the ASCII–text–only line printers of that time, so any support for printing options, as output quality or so, were not needed. Due to Unix being used only on mainframes in computing centers where only experts are operating the computers newbie–friendliness did not have a high priority, too.

Unfortunately LPD was used for a very long time, up to even nowadays, but printers changed a lot, they could print graphics, in color, on various paper types and sizes, for internal or presentation purposes, and so on. And printers got so cheap that many people have them at home.

To make use of modern printers most Unix applications describe the pages to print in PostScript and send this data to the printer spooler. As many printers do not understand PostScript, the spooler has to translate this into the printer's language. To do so, it calls GhostScript, a software PostScript interpreter running on the computer. GhostScript contains the printer drivers, compiled into its executable binary. Every driver knows the protocols of certain printer models. Due to the drivers being written by many different programmers, they have all very different options, to be set on the GhostScript command line or to be inserted into the PostScript code sent to GhostScript. In addition there are filter–style drivers: GhostScript generates a standard bitmap format and the driver as a separate process translates the bitmap format into the printer's format.

This makes manual configuring of printers a very complicated task. So the distribution manufacturers had to find a solution to make it possible for the end user to set up as many different printer models as possible. So systems like the RHS Printfilters or APS filters were created, but they have still many disadvantages:

- Every distribution did its own thing.
- Only one spooler was supported (Usually LPD).
- Not all and the newest drivers were supported.
- Options (as resolution, ...) can only be set by the administrator when he sets up the queue.
- Not all options were available.
- Difficult to add new drivers and printers.

In contrary to the older printer support database and integration systems, Foomatic currently has a database of more than 1000 printer models, nearly all known free software printer drivers (around 240) and the complete information how the drivers are executed and which options are available. Foomatic comes with scripts to generate Adobe–compliant PPD (*Postscript Printer Description*, describes capabilities and user–settable options of the printer) files or even complete print queues for all known free spoolers (CUPS, LPD, LPRng, GNUlpr, PPR, PDQ, CPS, and spooler–less printing). Foomatic–based print queues allow the user to adjust all options of the printer driver on a per–job (and even a per–page) basis. The development on linuxprinting.org is done independent of particular Linux distributions.

How did linuxprinting.org and Foomatic emerge?

The linuxprinting.org website has evolved from the Printing HOWTO, which Grant Taylor first wrote in 1992 and has maintained ever since. Starting in 1998, he began operating a little database of printer support information, and that little list has now grown into the largest repository of free software printing–related information around.

CUPS makes its appearance

In June 1999, the new CUPS printing system did undergo its first public release. Besides the innovative *printer browsing* features of its spooler, it contained a Ghostscript–based PostScript interpreter (the "`pstoraster`" filter). PostScript printers were supported by the usage of their original PPD files (which are part of the Windows/MacOS drivers).

How CUPS uses the original PostScript PPD files

The CUPS server associates the appropriate PPD to each queue. It parses the PPD and extracts the user–available printing options for the related target printer. Clients get told the PPD options for the target printer by the server *on the fly*. Clients therefore don't need the PPD installed locally: instead they receive the PPD options as a simple list of choices from which they can build a commandline. GUI tools can translate the list of printer options into nice dialogs. The client just selects the target printer and sends the desired print options as command line parameters to the CUPS server, along with the data to be printed (in most cases PostScript). The CUPS server inserts then, based on the descriptions of the individual options in the PPD, the PostScript commands needed to execute the option settings into the PostScript which would go to the printer (or rather, the printer's PostScript interpreter). For the first time a UNIX printing system was there, which supported the same printing options as known in the Windows world, by simply using the same PPDs that were created by the manufacturers for each of their PostSript models.

How CUPS extends use of PPDs to the non–PostScript printer world

However, CUPS doesn't limit its PPD support to PostScript printers and their RIPs (*Raster Image Processors*, here PostScript interpreters) only. It rather extends it to non–PostScript models too. Of course, you couldn't get a PPD for non–PostScript consumer inkjets at the time. And PPDs were never intended for non–PostScript printers either.

But the CUPS developers translated the printing options (and related low–level printer commands) available for some popular inkjets, laser printers, and dot matrix printers into a PPD–conforming syntax and can thusly handle those printers inside the same framework as the real PostScript models. CUPS adds one simple line to the PPD ("`*cupsFilter...`"), which tells the CUPS server how to handle the data to be printed (for example calling a software RIP). For the clients, any CUPS printer is a *PostScript* printer. To them it is not relevant, *where* the RIP is located: the PostScript may be interpreted by the device itself or by a software RIP on the server. They want their paper coming out of the printer as required.

Not enough supported printers...

CUPS provides a brilliant concept for handling print options for all printer types through PPDs –– however it doesn't deliver many PPDs. It ships a few sample PPD files, which are generic enough to support a few hundred PCL laser (HP LaserJet compatible), PCL inkjet (HP DeskJet compatible), Epson inkjet, and dot matrix printers –– but the very *generic* part of the picture made them stop short of handling the very specific features of each model. And it didn't support at all many models which were supported by GhostScript.

Grant initially was no great fan of CUPS. However, the way CUPS handled the files to be printed with the help of PPDs, ignited his idea to design a mechanism which would automatically generate CUPS–compatible PPDs from the contents of his database.

CUPS–O–Matic and Foomatic –– the first incarnations

If CUPS could guide the print data through its "`pstoraster`" filter (CUPS's own RIP, in that time based on GNU GhostScript 5.50), couldn't it also be guided through the system–installed GhostScript? Of course, CUPS' modular design not only allows for, but even invites developers to contribute additional filters. Grant didn't want to add an *additional* filter. He wanted to get all the printers running with traditional GhostScript–based spoolers to work with CUPS too. For many drivers his database contained the complicated GhostScript command line parameters, which made the GhostScript filters (=*devices*) and models work. This info could surely go into a PPD–like file, too, describing a driver setup for a GhostScript filter/printer model combo.

In early 2000, he threw out the driver half of his database and designed a new driver information scheme which gives users of many printing systems vastly enhanced support for the use of free software drivers. The first version of *CUPS–O–Matic* appeared. It was an online–generator for PPD–files to be used with CUPS. You could use a browser, surf to the CUPS–O–Matic page, select your model, choose an appropriate driver (or Ghostscript device), click "OK" and generate on the fly a PPD file. It needed an additional "`cupsomatic`" Perl script installed as a CUPS filter, which did the converting from PostScript to the printer's native data format with the help of GhostScript. It replaces CUPS's own RIP.

PDQ–O–Matic next...

Soon after, he also had the first automatic configuration generator for *PDQ* in place, which long after remained his personal favorite printing system recommended to end users (PDQ however cannot be used as a print server as it has no daemon listening for incoming jobs). Also an *LPD–O–Matic* got added and so the database got the name *Foomatic*.

Creation of "Linuxprinting.org"

In June 2000, Grant moved the whole thing out from his personal area at picante.com to the newly acquired domain linuxprinting.org, which better reflects this site's purpose (and which is certainly easier to remember!). Later that year, he bought a shiny new server and collocated the thing to better handle the traffic.

MandrakeSoft goes CUPS!

In August 2000, Till Kamppeter was employed by MandrakeSoft in Paris due to his XPP project. His task was switching from LPD to CUPS as default printing system in Mandrake Linux 7.2.

To make this reality without loosing support for any printer which was supported under Mandrake Linux 7.1, he made use of the Foomatic database to integrate the good old GhostScript printer drivers in the CUPS system. He got write access to the database from Grant so that he could enter the execution data of all the drivers, which is essential for the database to generate the PPD files which CUPS needs to support all printer/driver pairs.

The Database grows up

This way the database was not only a collection of user reports how well printers are supported by free software, but also a powerful tool to configure printers under various spoolers. In contrary to other printer configuration database systems (as RHS Printfilters and APS filters) the Foomatic system offered full support to all user–settable options of all free software printer drivers (to obtain this, Till had often to consult the driver's source code due to lack of driver documentation) and supported three spoolers (CUPS, LPD and alike, PDQ).

In March 2001, Red Hat 7.1 came out as the second distribution using Foomatic data for their new printer setup utility "printconf" and dropping their RHS Printfilters which were the most used printer setup database before. Red Hat was using LPRng as printing system that time.

From Postgres to XML

Also in March 2001, the original Postgres–based system was thrown out and replaced with the new XML version of the Foomatic configuration and filter system. This made it possible to download the complete Foomatic database and use it on one's local machine. In theory this was possible, but the engine to generate the spooler–specific configuration files from the printer, driver, and option XML files was very slow and memory consuming (around 150 MB). In addition, it needed something like 10 Perl libraries for the XML handling, which made it difficult for inexperienced users to install Foomatic. This was version 1.1 of Foomatic.

In July 2001, Grant gave full administration access to Till so that he could maintain the system. Grant had not much time for it any more, due to his work at a new start–up company.

C instead of Perl: huge speed improvements for Foomatic

By the end of August 2001, Till introduced a C program into Foomatic to accelerate the generation of the configuration files substantially, which allowed for the first time to compute the files on–demand from a local XML database instead of shipping pre–compiled files for all printer–driver combos. He also did the full implementation of the "`foomatic-configure`" and "`foomatic-printjob`" scripts which provide printer configuration and print job handling interfaces which are independent of the spooler actually used.

In October 2001, Mandrake Linux 8.1 came out as the first distribution supporting three spoolers (CUPS, LPRng, and PDQ) equally, with the help of the Foomatic system and spooler–specific configuration files being computed on demand. This was done through a new version of "printerdrake" (Mandrake's printer setup tool) vastly improved by Till.

April 2002: A second C program added by Till liberated Foomatic from needing this big amount of Perl libraries. The libxml–based C program reads XML files and puts out Perl data structures which can be read directly by Perl programs without special Perl libraries. This way the installation of Foomatic got much easier, and the web site got faster. This is the initial work leading to version 2.0 of Foomatic.

Gathering more support for Foomatic and the Database

June 2002: Till met printing developers of Red Hat and SuSE on a Foomatic workshop which he has given on the LinuxTag 2002 in Karlsruhe in Germany. They joined the Foomatic developer team and a new design for Foomatic where all spoolers use PPD files as printer/driver description files was under discussion.

July 2002: Till released the version 2.0.0 of Foomatic to open a stable branch and to let the development of the new PPD–centric Foomatic happen in a development branch (version 2.9.x) to approach version 3.0.

Now Foomatic has evolved into to an unofficial standard: It serves as the printer/driver capabilities database and driver/spooler integration system for the GNU/Linux distributions Mandrake, Red Hat, Conectiva, Debian, and others. SuSE was participating in the development of Foomatic 3.0.x, so they prepared for using Foomatic. In addition, around 5000 people every day visited the linuxprinting.org web site.

September 2002: With Mac OS X Apple switched to a Unix–based operating system, with a kernel derived from FreeBSD, but with their own, proprietary desktop environment. From version 10.2 on Apple used CUPS as the printing system. Tyler Blessing started to make Mac OS X packages of ESP GhostScript and the most important free software printer drivers. So many printers for which there are no manufacturer–supplied printer drivers got working under Mac OS X. Since then, up to the end of the year the number of visitors per day doubled to around 10000.

December 2002: After longer e–mail discussion on the Foomatic developer forum, especially with Johannes Meixner from SuSE, Till announced the first release in the Foomatic 2.9.x development series, Foomatic 2.9.0. This was the first version of Foomatic, where PPD files and the universal "`foomatic-rip`" filter are used with all spoolers. "`foomatic-rip`" makes it also possible for the first time that manufacturer–supplied PPD files of PostScript printers can be used with every spooler.

April 2003: Mandrake Linux 9.1 is the first distribution using Foomatic 3.0.x.

May 2003: Foomatic 3.0.0 is released and replaces the old Foomatic 2.0.x series. Besides completely *Adobe−compliant PPD files* and "`foomatic-rip`" there are many new features, as manufacturer−PPD support for all spoolers, custom page sizes, applying options to selected pages, composite options, ...

The Foomatic system as it is now

The XML database

The core of Foomatic since version 1.1 is an XML database containing one file for every printer, one file for every driver, and one file for every adjustable option. These files contain all the needed information about the printer and driver capabilities for both forming the database entry pages on linuxprinting.org and providing the information about the driver's command lines and their options.

The files contain:

- *Printers*: Contain make, model, mechanism type (inkjet, laser, ...), maximum resolution, color/grayscale, how well they work under free software, comments, information about consumables, ...
- *Drivers*: Contain name, type, command line prototype, comments, list of supported printers, ...
- *Options*: Contains name, type, for which printer(s) and driver(s) they are, default setting (depending on printer/driver), list/range of possible settings, names of the settings, for which printer(s)/driver(s) each setting is valid, strings to insert into the driver's command line.

How does the database work? – A small example

To explain how the database is is structured, we assume that we have a very small database, consisting only of the entries shown in Fig. 1: 4 printers, 3 drivers, and 2 options. The real database naturally has many more entries, especially every driver *must* have a "PageSize" option to make the spoolers working correctly. The information contained in the XML files you see in the colored boxes, the white boxes under the printer entries show what can be derived from the information stored in the database.

At first let's see the "ljet4" driver, a driver for printers which understand PCL 5e and with maximum resolutions up to 600 dpi. Its printer list contains all printers which understand PCL 5e: HP LaserJet 4, HP LaserJet 2100, and Epson EPL−5900. The Epson Stylus C80 does not understand PCL.

The "pxlmono" driver serves for PCL 6 printers with up to 1200 dpi, in our case the HP LaserJet 2100 and the Epson EPL−5900.

The "gimp−print" driver supports all printers in our example because it generates various languages, including PCL 5e for our lasers and ESC/P 2 for the Epson Stylus C80.

Note that the printers and the drivers are not associated by the printer's languages. A printer is only considered as supported by a driver when it is in the printer list of the appropriate driver

entry.

All information about which options and possible option settings are available for a certain printer/driver pair are stored in the option XML files as the so–called constraints. Constraints can qualify or disqualify options or settings for a certain manufacturer, model, and/or driver. The "Resolution" option in our example has constraints that qualify it for the "ljet4", "pxlmono", and "gimp–print" drivers, so it is available for all drivers in our example. Due to no printer restrictions made here, all printers/driver combos with one of the three shown drivers will have the "Resolution" option here.

There are three choices "600 dpi", "1200 dpi", and "720 dpi". If they had no constraints they applied for all printers/driver combos, but as laser printers and so also dedicated laser printer drivers only having multiples of 300 dpi as resolutions and Epson inkjets only having multiples of 360 dpi, we have defined constraints for the settings, so that the user gets only valid resolutions presented for his printer. The "600 dpi" resolution is qualified for the "pxlmono" and "ljet4" drivers for arbitrary printers. These drivers support only lasers and all lasers in our example do 600 dpi. For 600 dpi with "gimp–print" we have additional restrictions to our three lasers because the C80 works with GIMP–Print, but not with 600 dpi. For the "1200 dpi" we had to impose the constraint to only "pxlmono". The other two drivers are only PCL 5e drivers and so they drive laser printers only up to 600 dpi. And as all lasers not supporting 720 dpi, the "720 dpi" choice is restricted to exclusively being valid for the C80, independent which driver is used.

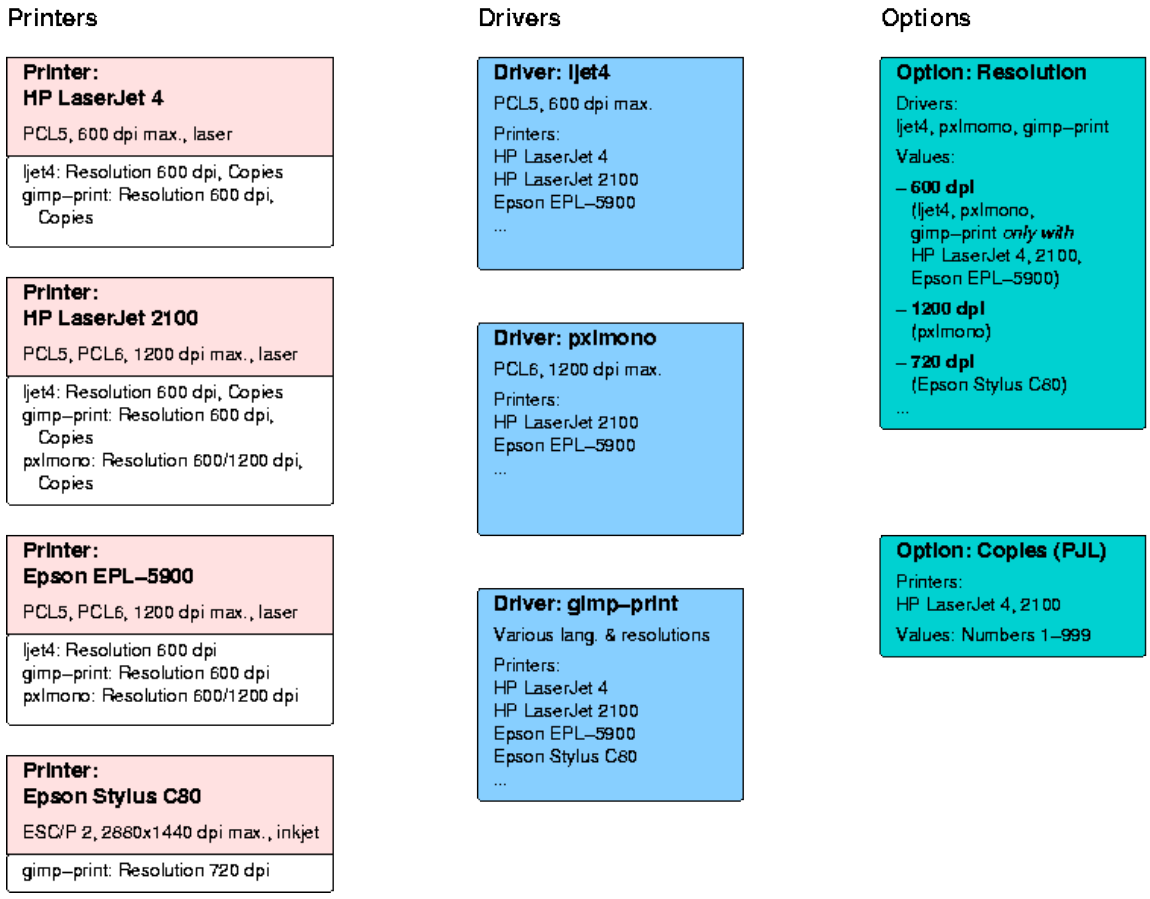

Fig. 1: An example to explain the structure of the database

The second option in our example is the numerical option "Copies", it allows integer numbers from 1 to 999 as settings, but the file only contains the range, not setting entries for every possible number. Therefore no constraints can be applied to the settings. One has to define various "Copies" option entries when there would be printers with different maximum numbers of copies.

Our "Copies" option is not an option of any of the drivers, it is an option of the printers which can be set by sending a so–called PJL (Printer Job Language) command to the printer before sending the job data itself. This facility is available in nearly every PostScript or PCL laser printer and the commands are the same, independent whether the job itself is sent in PostScript, PCL 5, or PCL 6 (PostScript printers usually understand also PCL). So the commands are independent of the printer driver used for the job itself. Therefore PJL options have only printer constraints, in our example the candidates understanding PJL and so also the "Copies" option are the two HP lasers, the LaserJet 4 and 2100.

What is done to set up a print queue with this data?

The first thing needed is a PPD file for the desired printer/driver combo. The principle of obtaining this is the same for both generating it on the linuxprinting.org web site or with a local copy of Foomatic.

At first a combo XML file for the chosen printer/driver combo is generated. After checking that the printer is in the driver's printer list a C program ("`foomatic-combo-xml.c`") parses the printer entry, the driver entry and then all option entries. The printer entry is written entirely into the combo XML file, from the driver entry all except the printer list gets into the combo file and from the options only the ones which are valid for the printer/driver combo, with all constraints and invalid settings taken out. The C program does not use any XML libraries and loads only one source XML file at a time, parses it sequentially, and writes the read data directly into the combo XML file if it is needed. This makes the process very fast and less memory–consuming. The combo XML file is a spooler–independent XML representation of the capabilities of the printer/driver combo and of how one prints PostScript files with it.

Now the combo XML file is parsed by another C program ("`foomatic-perl-data.c`") which uses `libxml2` from `www.xmlsoft.org` to generate a Perl data structure from the XML file. This is done by a C program because there is one standard XML library with which the process can be easily and fastly done. Doing this in Perl is much more complicated. This still spooler–independent Perl data structure contains most of the data which the combo XML file provides, especially all the information about how to execute the driver. This information goes into the PPD file so that the spooler, client machines, and the "`foomatic-rip`" filter script know about how to use the driver and how to apply the user–supplied option settings.

Now a spooler–independent, completely Adobe–compliant PPD file is built. In contrary to the former Foomatic 2.0.x in this PPD file no information is hidden in comment lines. Everything which cannot be defined with standard keywords (as for example the GhostScript command line) is defined with special keywords beginning with "`*Foomatic...`". The Adobe specification allows custom keywords and requires applications who do not know a certain keyword, to ignore it. This PPD file is the file carrying all necessary information about the printer and the driver in the configuration of the print queue. Either the user downloads it from linuxprinting.org or the "`foomatic-configure`" Perl script of a local Foomatic installation creates it when setting up the print queue.

In addition to the PPD file the universal filter script "foomatic-rip" has to be installed. It will be called by the spooler to translate the incoming PostScript job data into the printer's native language.

"foomatic-rip" is a Perl script and reads at first the printer and driver information from the PPD file. For that no extra Perl libraries or C programs are needed. The "foomatic-rip" gets the option settings given by the user either from the spooler via command line option/environment variables or embedded in the PostScript job data. With this information the "foomatic-rip" builds the appropriate GhostScript command line and executes it. It also inserts settings into the job data, either PostScript or PJL commands.

How does printing with Foomatic work?

Fig. 2 shows a diagram of the data flow when Foomatic is used for printing. On the system there is already a spooler (dark cyan box) and GhostScript with a driver (cyan box). A print queue is set up using the linuxprinting.org web site or a local Foomatic installation (light yellow box). For this a PPD file and "foomatic-rip" is installed (yellow boxes).

The data to be printed goes usually from the application program (light red) through a printing frontend (blue) to the spooler (dark cyan), and from there through the filter (yellow) and GhostScript with driver (cyan) to the printer.

When one executes the printing command in an application, it usually produces PostScript as output and sends it to a printing frontend, normally the command line frontend "lpr". The user can often modify the printing command to choose a printer and to set options, or to use another command than "lpr". An "lpr" command line to print with a resolution of 1200 dpi on the printer "lj" can look like this:

```
lpr -P lj -o Resolution=1200 file.ps
```

The option settings specified on the command line (red line) accompany the PostScript data (blue line) when the job goes to the spooler.

If the user chooses a graphical printing frontend ("kprinter", "xpp", "gtklp", "gpr", ...) as the printing command in his application, a window pops up and shows a list of available printers and gives the possibility to open a dialog with printer-specific options. The frontend must get the information about the printers and options somehow (brown lines). In case of CUPS as the spooler all frontends poll this information from the CUPS daemon (black lines) and CUPS itself takes the printer option information from the PPD file of the appropriate print queue (brown line). In case of LPD, LPRng, or GNUlpr being the spooler "gpr" reads the Foomatic-generated PPD file directly (brown line). "gpr" is a special GUI frontend: It uses PPD files and stuffs all option settings into the PostScript data (magenta line), so it works with any spooler and also if there are different spoolers on the client and on the server. "kprinter" calls "lpr" with the option settings on the command line (blue and red lines).

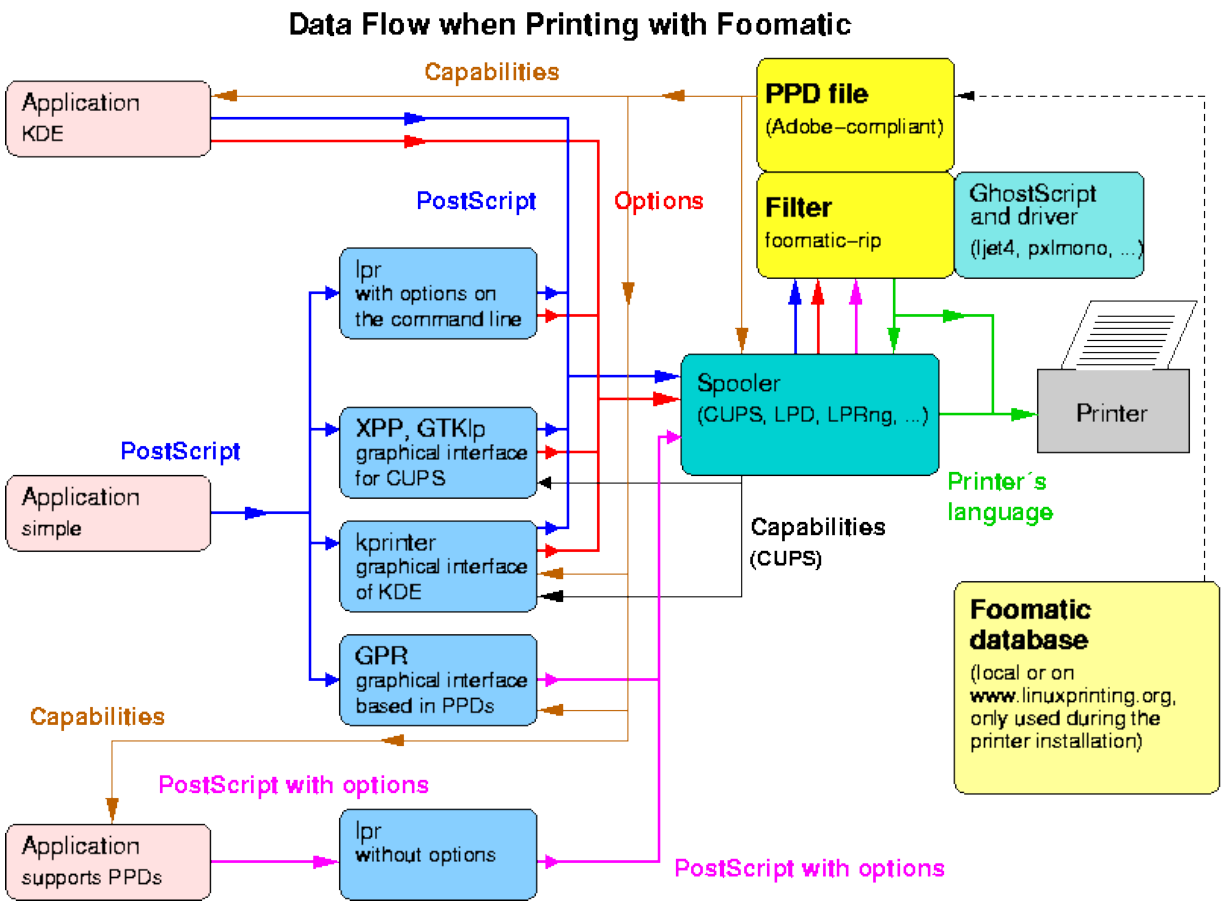

Fig. 2: Data flow when printing with Foomatic

KDE applications use "`kprinter`" as their printing dialog, so they behave as "`kprinter`".

In PPD–aware applications as Star Office, OpenOffice.org, the GIMP, or Windows/Mac clients using a PostScript driver, one assigns a PPD file (in our case the Foomatic–generated one) to every print queue and the application gets the printer information from that file. The option settings are all embedded in the PostScript data (magenta line), as with "`gpr`".

The spooler calls "`foomatic-rip`" and hands over all the PostScript and option settings data to it. Then "`foomatic-rip`" parses the PPD file, builds the GhostScript command line according to the user's option settings, and executes it. The resulting data in the printer's language (green line) goes finally to the printer.

The structure of the XML database

Here we want to get deeper into the structure of the XML data files. Therefore we explain one example printer, driver, and option XML file. The following text is mainly taken from the `README` file of the "foomatic–db–engine" package.

An option entry: "PageSize" (`db/source/opt/2.xml`)

Every option exists independently from printers or drivers, because they might apply to arbitrary combinations of printers and/or drivers. In practice, some drivers have wholly unique options

("gimp−print"/"stp", for example), while others (lots of generic basic Ghostscript drivers, for example) share some options.

```
<option type="enum" id="opt/2">
```

Options are of a type "enum", "bool", "int", "float", "string", or "password", options have an ID. The id is also the filename.

The shortname is a spaceless short name for the thing. It must not contain "/" or ":" (otherwise it will not be handled correctly in PPD files). It should be one of the standard Adobe PPD option names if appropriate

```
  <arg_shortname>
```

Various things here, and all <comments>, are internationalized. They take the usual posit locale codes in the form ox[shy], where ox is a two−letter is language code, and YY is two−letter country code to distinguish differing national dialects.

Generally the national dialects won't be very common or necessary here. The backends currently require that <en> content be provided.

```
   <en>PageSize</en><!-- backends only know <en> shortnames! -->
  </arg_shortname>
```

The longname is a short phrase describing the thing in more detail, GUI tools usually show longnames

```
  <arg_longname>
   <en>Page Size</en>
  </arg_longname>
```

The comments are used to form documentation. In theory these can become man pages or the like.

```
  <!-- A multilingual <comments> block can appear here, too;
       it should be treated as documentation for the user. -->
```

The execution section describe how the backend should execute this option. The order and spot apply to the *driver*'s prototype for <arg_substitution /> (once called commandline) style options, or just the order applies for <arg_postscript /> and <arg_pjl /> options. The order and the <arg_section> go into the "*OrderDependency" line of the appropriate option entry in the PPD file, for this example one would get

```
   *OrderDependency: 100 DocumentSetup *PageSize
```

When no <arg_section> is given, "AnySetup" is used as a default.

For <arg_substitution /> options the <arg_proto> is inserted into the driver's command line, at the spot (e. g. "%A") whose letter is given between the <arg_spot>...</arg_spot> tags, the <arg_proto> of an <arg_postscript /> option is a snippet of PostScript code which is inserted into the PostScript data stream of the job, for DSC−conforming PostScript into the section specified with <arg_section>, otherwise in the beginning. The <arg_proto> lines of <arg_pjl /> options are PJL commands which are sent to the printer before the output of the driver's command line is

sent. Because this only works reliably when the driver output does not have its own PJL command header, these options are ignored when the driver's XML file is marked with a <nopjl /> tag in its <execution> section. Drivers which produce their own PJL and therefore are marked with <nopjl /> are for example "hpijs" and "hl1250". There is also the <arg_composite /> execution style for composite options, see the "Composite Options" section below. The user's value gets put into the <arg_proto>'s %s location.

The <arg_group>...</arg_group> tags put the option into the PPD option group named here. In many PPD–based GUIs ("kprinter", "xpp", OpenOffice.org, ...) every group is shown as a tab or a tree branch containing the member options of this group. You can also specify subgroups. Then you have to use a "group path" similar to directory paths, with the group and subgroup names separated by slashes (<arg_group>General/Paper</arg_group> is the "Paper" subgroup in the "General" group). Subgroups are not recommended as there is no GUI supporting them. If an option is member of a composite option (See "Composite Options" section below), the <arg_group>...</arg_group> tags will be ignored.

```
<arg_execution>
 <arg_group>General</arg_group>
 <arg_order>100</arg_order>
 <arg_section>DocumentSetup</arg_section>
 <arg_spot>Z</arg_spot> <arg_postscript />
 <arg_proto><</PageSize[%s]/ImagingBBox null>>setpagedevice</arg_proto>
 </arg_execution>
```

The constraints define what printer/driver combinations this option applies to. The *most specific* constraint rules the day; it's "sense" says whether or not the option is "in". The winning constraint also provides the default value used when this option applies to that printer and driver.

Constraint elements are: driver, make, model. The driver is the driver name, or not present to apply to any driver. The make is the printer make, or not present to apply to any printer make. The model is the driver model, or not present to apply to any printer. Instead of make/model, you can also specify <printer>id</printer>.

IMPORTANT: The make and model must match the one in the printer xml definition, and everywhere else in the other options. One needs to write a utility to change printer names sensibly.

- It is illegal to have a model with no make.
- It is illegal to have none of make/model/driver.
- It is illegal to have *no* constraints, or at least such options are never used.

For enum options, the defval is the id of the enum_val that is the default. For other option types, it is the actual default value (i. e, a number, or 1 or 0 for boolean, etc).

```
<constraints>
   <constraint sense="true">
    <driver>sj48</driver>
    <arg_defval>ev/1</arg_defval>
   </constraint>
   <constraint sense="true">
    <driver>r4081</driver>
    <arg_defval>ev/1</arg_defval>
   </constraint>
```

(A gaillion constraints skipped)

```
</constraints>
<enum_vals>
 <enum_val id="ev/1">
  <ev_longname>
   <en>US Letter</en>
  </ev_longname>
  <!-- A multilingual <comments> block can appear here, too;
       it should be treated as documentation for the user. -->
  <ev_shortname>
   <en>Letter</en>
   <!-- Until someone tells me how to learn the user locale in
        backends, the shortname must be monolingual in <en>! -->
  </ev_shortname>
```

If present, the driverval is what gets substituted in for the %s in the option's prototype. This way the user–visible stuff can be anything.

```
   <ev_driverval>612 792</ev_driverval>
```

This enum_val has no constraints. It *is* OK for enum_vals to have no constraints; they are assumed to apply unless constrained otherwise.

```
</enum_val>
<enum_val id="ev/115">
 <ev_longname>
  <en>A3</en>
 </ev_longname>
 <!-- A multilingual <comments> block can appear here, too;
      it should be treated as documentation for the user. -->
 <ev_shortname>
  <en>A3</en>
  <!-- Until someone tells me how to learn the user locale in
       backends, the shortname must be monolingual in <en>! -->
 </ev_shortname>
 <ev_driverval>842 1191</ev_driverval>
```

Here are some example constraints for an enum_val. The A3 size paper doesn't fit on lots of printers, so there are various constraints to make the right thing happen.

```
 <constraints>
  <constraint sense="true">
   <driver>ml85p</driver>
   <arg_defval>na</arg_defval>
  </constraint>
  <constraint sense="true">
   <make>HP</make>
   <model>DeskJet 1000C</model>
   <driver>pnm2ppa</driver>
   <arg_defval>na</arg_defval>
  </constraint>
  <constraint sense="false">
   <make>HP</make>
   <model>DeskJet 820C</model>
   <driver>pnm2ppa</driver>
   <arg_defval>na</arg_defval>
  </constraint>
```

(lots more...)

```
    </constraints>
   </enum_val>
  </enum_vals>
</option>
```

To allow custom page sizes to be used one has add a choice with the <ev_shortname> being "Custom" to the "PageSize" option (example below). This choice will be treated as the custom page size. When the user selects this choice, he has to provide the width and the height of the page in addition. These values are converted into PostScript points (1/72 inches) and inserted into placeholders in the <ev_driverval> of this choice. The <ev_driverval> should contain a placeholder "%0" for the page width and "%1" for the page height. Alternatively the <ev_driverval> can contain two zeros ("0") from which the first will be replaced by the page width and the second by the page height. Then one gets Adobe–compliant entries for the custom page size in the PPD files and one can set a custom page size with the following commands:

CUPS:	`lpr -P huge -o PageSize=Custom.500x750cm bigposter.ps`
LPRng:	`lpr -P huge -Z PageSize=Custom.500x750cm bigposter.ps`
GNUlpr:	`lpr -P huge -o PageSize=Custom.500x750cm bigposter.ps`
LPD:	`lpr -P huge -JPageSize=Custom.500x750cm bigposter.ps`
PPR (RIP):	`ppr -P huge -F "*PageSize Custom" --ripopts 500x750cm bigposter.ps`
PPR (Interf.):	`ppr -P huge -F "*PageSize Custom" -i 500x750cm bigposter.ps`
PDQ:	`pdq -P huge -oPageSize_Custom -aPageWidth=500 -aPageHeight=750 -oPageSizeUnit_cm bigposter.ps`
No spooler:	`foomatic-rip -P huge -o PageSize=Custom.500x750cm bigposter.ps`

Here is an example for a custom page size setting:

```
<enum_val id="ev/PageSize-Custom">
 <ev_longname>
  <en>Custom size</en>
 </ev_longname>
 <!-- A multilingual <comments> block can appear here, too;
      it should be treated as documentation for the user. -->
 <ev_shortname>
  <en>Custom</en>
  <!-- Until someone tells me how to learn the user locale in
       backends, the shortname must be monolingual in <en>! -->
 </ev_shortname>
 <ev_driverval>%0 %1</ev_driverval>
</enum_val>
```

The entry

```
 <ev_driverval>0 0</ev_driverval>
```

would have the same effect as the <ev_driverval> of the example.

For numerical (int, float) and bool options there is no <enum_vals> section. Instead of this section numerical options have tags to specify minimum and maximum value:

```
<arg_max>10.0</arg_max>
<arg_min>0.0</arg_min>
```

For the %s in the <arg_proto> a number, either the user's choice when he has specified this option or the default value is inserted. Only numbers between the minimum and the maximum and in case of int options only integer numbers are allowed.

Bool options can be set or not be set. There <arg_proto> will be inserted if they are set, nothing if they are not set. A %s in the <arg_proto> is not allowed, there is nothing to insert for it. As <arg_defval> in the option's constraints one can use 0 for not setting the option by default or 1 for setting it by default.

Bool options need the specification of a name for the case when they are not set. This will be used by GUIs and in PPD files:

```
<arg_shortname_false>
  <en>CorrectBlack</en><!-- Backends only know <en> shortnames! -->
</arg_shortname_false>
```

This name should not contain spaces, ":", or "/".

See below for string and password options.

A printer entry: HP LaserJet 4000 (db/source/printer/HP-LaserJet_4000.xml)

The printer file contains information specific to a particular printer.

```
<printer id="printer/HP-LaserJet_4000">
```

Make and model are not internationalized. There will eventually be an "alias" mechanism, but the need is different.

```
<make>HP</make>
<model>LaserJet 4000</model>
```

According to the Adobe specifications for PPD files every PPD file must contain a unique DOS–compatible file name (the "*PCFileName"), a file name with an up to 8 characters log base name and an up to 3 characters long extension, and upper and lower case letters being considered as equal. As every PPD file is for a printer/driver combo, we let the first 6 characters being provided by the printer entry:

```
<pcmodel>HPLJ4K</pcmodel>
```

The first two characters should be the manufacturer prefix as listed in Appendix D of Adobe's "PostScript Printer Description (PPD) File Format Specification Version 4.3".

Various stuff about the machine

```
<mechanism>
```

Printer types can be <laser />, <led />, <inkjet />, <dotmatrix />, <impact />, <sublimation />, <transfer />. Other types we have to add to the CGI script on linuxprinting.org to make the web interface displaying them properly.

```
  <laser/>
```

At some point we can make color be less of a boolean flag and more of a section full of goodies.

```
  <!--not "color"-->
  <resolution>
```

In theory this is a list. In practice We've only got one per printer which is the maximum resolution the manufacturer claims for this printer.

```
    <dpi>
      <x>1200</x>
      <y>1200</y>
    </dpi>
  </resolution>

  <consumables>
```

Information about ink, drums, etc. The comments are supposed to be qualitative ("Separate drum and toner cartridges")

```
    <comments>
      <en>toner</en>
    </comments>
```

There should be <partno>12A1975</partno> elements with manufacturer part numbers for the various carts, etc it takes. Then one could have a price watcher thingy like there is now for the printers.

```
    <!--one or more "partno" elements.-->
  </consumables>
</mechanism>

<url>http://www.pandi.hp.com/pandi-db/prod_info.show?model=C4118A&name=LaserJet4000</url>
```

The lang section. In practice this will be only minimally useful:

- Backends can pstops the ps down a level if needed
- Backends know if pjl options apply
- Backends can know if "quick text" will work

Commonly used language tags: <pcl level="x" />, <escp2 />, <proprietary />

```
<lang>
  <postscript level="2">
  <!--unknown ppd filename "ppd"--></postscript>
  <pjl/>
  <text>
    <charset>us-ascii</charset>
  </text>
```

```
  </lang>
```

The autodetection stuff

```
  <autodetect>
```

There are three ways to auto–detect a printer, via the parallel port (<parallel>...</parallel>), the USB (<usb>...</usb>), or SNMP (TCP/Socket–connected printer, <snmp>...</snmp>). Through these interfaces the printers report back an IEEE–1284–compliant ID string from which the fields "MFG" (<manufacturer>...</manufacturer>), "MDL" (<model>...</model>), "DES" (<description>...</description>), and "CMD" (<commandset>...</commandset>) are used. The string itself can be put between <ieee1284>...</ieee1284> tags, but all items which are not constant for all printers of this model, as the serial number ("SERN:...;") and the device status ("VSTATUS:...;") have to be removed here. As the ID string is usually the same for all detection methods, one can put the entries between <general>...</general> tags, then the <parallel>...</parallel>, <usb>...</usb>, and <snmp>...</snmp> are only used for differences to the data between the <general>...</general> tags. A complete entry could look like:

```
  <autodetect>
    <general>
      <ieee1284>MFG:HEWLETT-PACKARD;MDL:DESKJET 600;CMD:MLC,PCL,PML;CLASS:PRINTER;DESCRI
      <commandset>MLC,PCL,PML</commandset>
      <description>Hewlett-Packard DeskJet 600</description>
      <manufacturer>HEWLETT-PACKARD</manufacturer>
      <model>DESKJET 600</model>
    </general>
  </autodetect>
```

On Linux you find this info for the parallel ports (`/dev/lp<N>`, `<N> = 0, 1, 2, ...`) in the files

```
   /proc/sys/dev/parport/parport0/autoprobe*
```

for the USB under Linux it is more complicated, easiest is to use a little Perl script, called "getusbprinterid.pl":

```
#!/usr/bin/perl

open FILE, "$ARGV[0]" or die;

my $result;
# Calculation of IOCTL function 0x84005001 (to get device ID string):
# len = 1024
# IOCNR_GET_DEVICE_ID = 1
# LPIOC_GET_DEVICE_ID(len) = _IOC(_IOC_READ, 'P', IOCNR_GET_DEVICE_ID, len)
# _IOC(), _IOC_READ as defined in /usr/include/asm/ioctl.h

ioctl(FILE, 0x84005001, $result) or die;
close FILE;

# Cut resulting string to its real length
my $length = ord(substr($result, 1, 1)) + (ord(substr($result, 0, 1)) << 8);
$result = substr($result, 2, $length-2);

# Remove non-printable characters
$result =~ tr/[\x0-\x1f]/\./;
print "$result\n";
```

Running the program with "`./getusbprinterid.pl /dev/usb/lp0`" returns the ID string of the device on `/dev/usb/lp0`.

```
    <!--no known parport probe information-->
  </autodetect>
```

Our grading system. It's US−style letter grades A, B, D, and F, which the website shows as "Perfectly", "Mostly", "Partially" and "Paperweight". *THERE IS NO "C"!!!*

```
  <functionality>A</functionality>
```

Arguably, the scores should live with the printer/driver association and not on the printer, but then it's a big hassle to figure out if a printer works... So the score is the one reached with the driver working best, the "recommended" driver.

There's a spot for this "recommended" driver, usually the driver which gives the maximum output quality. It is for user information on the web site, but newbie−friendly printer setup GUIs should use it, too. Unfortunately, only "printerdrake" of Mandrake Linux makes use of it.

```
  <driver>Postscript</driver>
```

The <unverified /> tag was on all printer entries which were formerly entered by visitors using the web printer input interface as the database was still PostGreSQL−driven.

```
  <!--not "unverified"-->
```

If there is a web site with additional interesting info about this printer, it can be mentioned in the entry by putting it between <contrib_url>...</contrib_url> tags,

```
  <!--no "contrib_url"-->
```

The regular notes section. The allowed tags are: <p>,   and many other simple tags (<b>, <i>, <tt>, ...). Note that to distinguish what is XML and what is the embedded HTML, make the following replacements:

```
      <   -->   &lt;
      >   -->   &gt;
      "   -->   &quot;
      '   -->   &apos;
      &   -->   &amp;

  <comments>
    <en>
    I don&apos;t believe this:&lt;p&gt;

    &lt;i&gt;1200x1200 dpi only possible with Windows drivers,
    600x600 can be reached w/o particular software.
    The difference is visible, but only slightly, so
    the Functionality got &quot;Mostly&quot;&lt;p&gt;&lt;/i&gt;&lt;p&gt;

    Do the following:&lt;p&gt;

    Set the resolution on the front panel to &quot;Prores 1200&quot;, not
    to &quot;Fastres 1200&quot;. When you use CUPS with HPs PPD file, turn
```

```
    off &amp;quot;Fastres 1200&amp;quot; in the printer configuration
    options.&amp;lt;p&amp;gt;

    Try the generic PostScript PPD file which comes with KUPS 1.0 or newer.
    </en>
  </comments>
</printer>
```

A driver entry: "md2k" (db/source/driver/md2k.xml)

The driver files contain information about drivers. There are a few things, but the two biggies are the prototype and the printers list

```
<driver id="driver/md2k">
 <name>md2k</name>
```

According to the Adobe specifications for PPD files every PPD file must contain a unique DOS–compatible file name (the "*PCFileName"), a file name with an up to 8 characters log base name and an up to 3 characters long extension, and upper and lower case letters being considered as equal. As every PPD file is for a printer/driver combo, we let the last 2 characters being provided by the driver entry:

```
 <pcdriver>M2</pcdriver>
 <url>http://plaza26.mbn.or.jp/~higamasa/gdevmd2k/</url>
 <execution>
```

Driver types are:

`<ghostscript />`:	The driver code is compiled into GhostScript
`<filter />`:	The driver code is a separate executable, either a filter which converts generic bitmap output of GhostScript to the printer's language, a wrapper around GhostScript, or an IJS plug–in.
`<uniprint />`:	A uniprint driver, consisting of one or more .upp files for GhostScript.
`<postscript />`:	A driver which has PostScript also as output (for PostScript printers). It usually does not call GhostScript but only applies the user's option settings to the data stream. But GhostScript can be called here, too, as for downgrading to a lower PostScript level.

The driver type only provides information for the web pages, it is not used when generating config files for a spooler.

```
  <ghostscript />
```

The driver's <execution> section can also contain a

```
  <nopjl />
```

which suppresses the usage of PJL options (options which send PJL commands to the printer). This one does with drivers where the driver itself already produces a PJL header, the second one built by the PJL options would then be ignored by the printer, and so this kind of options does not make sense. Such drivers are for example "hpijs" and "hl1250".

The prototype defines what command the backends run to drive this printer. It must take postscript on stdin and generate "printer stuff" on stdout. Various %A, %B, etc substitution "spots" are specified; this is where substitution options will be placed.

```
  <prototype>gs -q -dBATCH -dSAFER -dQUIET -dNOPAUSE -sDEVICE=md2k%A%Z -sOutputFile=- -</proto
 </execution>
 <comments>
  <en>
    Part of the gdevmd2k-0.2a package by Shinya Umino.  The web page and
    documentation are in Japanese.
    &lt;a href=&quot;/clippings/MD5000-translation.txt&quot;&gt;Here&lt;/a&gt;
    is an English translation of the driver&apos;s web page, and &lt;a
    href=&quot;/clippings/alpsmd.txt&quot;&gt;here&lt;/a&gt; is the README from the
    driver package.
  </en>
 </comments>
```

The printer list is a simple list of printers that this driver works with. Historically, these "bits" were on the printer cgi form page, but now they're put here.

```
 <printers>
  <printer>
   <id>printer/Alps-MD-1000</id><!-- Alps MD-1000 -->
  </printer>
  <printer>
   <id>printer/Alps-MD-1300</id><!-- Alps MD-1300 -->
  </printer>
  <printer>
   <id>printer/Alps-MD-2000</id><!-- Alps MD-2000 -->
  </printer>
  <printer>
   <id>printer/Alps-MD-4000</id><!-- Alps MD-4000 -->
  </printer>
 </printers>
</driver>
```

In the printer list it is also possible to place comments specific to a certain printer/driver pair:

```
  <printer>
   <id>printer/HP-LaserJet_4050</id><!-- HP LaserJet 4050 -->
   <comments>
    <en>to 1200dpi</en>
   </comments>
  </printer>
```

Composite Options

This is an option type to make it easier for users to choose the best settings for a certain printing task, even if the driver has very many options. The idea is to have an enumerated choice option which does not directly modify something in the driver's command line but sets several of the other options.

One example is the "PrintoutMode" option which will be made available for all printer/driver combos which have at least one option regarding the printout quality or document type.

The possible choices should be the same for every printer and driver, so that users (especially newbies) can bring their printers in the right mode by choosing one easy to understand item from

a menu instead of having to switch several cryptic driver options. For now the choices are the following:

Command line	*GUI*	*Intention*
`Draft`	Draft	Very fast, ink/toner–saving printout
`Normal`	Normal	Quick standard quality printout
`High`	High Quality	High quality for plain paper
`VeryHigh`	Very High Quality	Highest quality for plain/inkjet paper
`Photo`	Photo	Highest quality for photo paper

These choices can also have one of the following modifiers:

Modifier	*Intention*
`.Gray`	Grayscale printing on a color printer
`.Mono`	Monochrome printing (no grayscales, black or white)

Examples:

Command line	*GUI*	*Comment*
`High.Gray`	High Quality Grayscale	
`Photo`	Photo	Color photos on color printer
`VeryHigh.Mono`	Very High Quality Monochrome	Really black text in highest quality on inkjet printer, not suitable for halftone images.
`Normal`	Normal	Standard color in 300/360 dpi on normal paper, grayscale on black–and–white printers

Not all choices/combinations of basic choices and modifiers must be present. Often modes are simply not available on certain printer/driver combos, as "Photo" on most lasers. It is highly recommended to have "Normal" available, though (and having this the default).

The GUI names can have additional remarks in parantheses, for example when manual intervention (other cartridge, photo paper) is needed.

To add such an option to the database, one only needs to add an option XML file like the one below into the db/source/opt directory of the database. The file db/source/opt/pcl3–PrintoutMode.xml could look like this:

```
<option type="enum" id="opt/pcl3-PrintoutMode">
  <!-- A multilingual <comments> block can appear here, too;
       it should be treated as documentation for the user. -->
  <arg_longname>
   <en>Printout Mode</en>
```

```
</arg_longname>
<arg_shortname>
 <en>PrintoutMode</en><!-- backends only know <en> shortnames! -->
</arg_shortname>
<arg_execution>
 <arg_order>10</arg_order>
 <arg_section>AnySetup</arg_section>
 <arg_spot>A</arg_spot>
 <arg_composite />
 <!-- <arg_proto></arg_proto> -->
</arg_execution>
<constraints>
   <constraint sense="true">
    <driver>pcl3</driver>
    <arg_defval>ev/pcl3-PrintoutMode-Normal</arg_defval>
   </constraint>
</constraints>
<enum_vals>
 <enum_val id="ev/pcl3-PrintoutMode-Draft">
  <ev_longname>
   <en>Draft</en>
  </ev_longname>
  <!-- A multilingual <comments> block can appear here, too;
       it should be treated as documentation for the user. -->
  <ev_shortname>
   <en>Draft</en>
   <!-- Until someone tells me how to learn the user locale in
        backends, the shortname must be monolingual in <en>! -->
  </ev_shortname>
  <ev_driverval>MediaType=Plain Resolution=150 Quality=Draft IntensityRendering=Halftones Pa
 </enum_val>
 <enum_val id="ev/pcl3-PrintoutMode-Normal">
  <ev_longname>
   <en>Normal</en>
  </ev_longname>
  <!-- A multilingual <comments> block can appear here, too;
       it should be treated as documentation for the user. -->
  <ev_shortname>
   <en>Normal</en>
   <!-- Until someone tells me how to learn the user locale in
        backends, the shortname must be monolingual in <en>! -->
  </ev_shortname>
  <ev_driverval>MediaType=Plain Resolution=300 Quality=Normal IntensityRendering=Halftones P
 </enum_val>
 <enum_val id="ev/pcl3-PrintoutMode-High">
  <ev_longname>
   <en>High</en>
  </ev_longname>
  <!-- A multilingual <comments> block can appear here, too;
       it should be treated as documentation for the user. -->
  <ev_shortname>
   <en>High</en>
   <!-- Until someone tells me how to learn the user locale in
        backends, the shortname must be monolingual in <en>! -->
  </ev_shortname>
  <ev_driverval>MediaType=Plain Resolution=600 Quality=Presentation IntensityRendering=Floyd
 </enum_val>
 <enum_val id="ev/pcl3-PrintoutMode-Photo">
  <ev_longname>
   <en>Photo (on photo paper)</en>
  </ev_longname>
  <!-- A multilingual <comments> block can appear here, too;
```


```
          it should be treated as documentation for the user. -->
     <ev_shortname>
      <en>Photo</en>
      <!-- Until someone tells me how to learn the user locale in
           backends, the shortname must be monolingual in <en>! -->
     </ev_shortname>
     <ev_driverval>MediaType=Premium Resolution=600 Quality=Presentation IntensityRenderi
    </enum_val>
   </enum_vals>
</option>
```

The shown option is only an example, it is neither in the CVS nore will it work with all printers which use the "pcl3" driver. You can paste it into a file (make the <ev_driverval>s being one line, the items separated by spaces) and copy it to db/source/opt/ to try it out.

The "<arg_composite />" tag for the execution style specifies it as a composite option. The <arg_spot> and <arg_proto> are meaningless in a composite option and the "<ev_driverval>"s contain a space−separated list of all settings of which the pre−made configuration represented by this choice consists. Every choice of the composite option must set *exactly the same* individual options. In no choice it is allowed to leave out one of them. These individual options are the member options of the composite option. Not all options of a driver/printer combo need to be member options of the composite option. It is not allowed to have one option being member of more than one composite option. The composite option must be an enumerated choice options, the member options must be enumerated choice or boolean.

It is enough to add a composite option as shown. The PPD generator (getppd() in lib/Foomatic/DB.pm, package "foomatic−db−engine") will take care of the rest. It will

- Order all member options into a group (PPD group, see "Option Grouping" below) named after the composite option.
- Add to every member option the choice "Controlled by '<name of the composite option>'" and make this choice the default. If this is chosen, the composite option will set the value for this member, depending on what value is chosen for the composite option. If the user chooses something else than "Controlled by '<name of the composite option>'" the member option does not obey the setting given by the composite option. So the advanced user can also set the member options individually.
- If necessary the <arg_order> and <arg_section> of the composite option is replaced by other values in the PPD file, so that the composite option will be stuffed into the PostScript data stream always before all its member options. Do not give "0" as the order number to any of the member options.

A composite option can also span only one (but not zero) member option. This is for example done with the "PrintoutMode" option of the HPIJS driver ("foomatic−db−hpijs" package). This driver has only one option for setting resolution and quality, but this options has sometimes many choices with rather cryptic names. The "PrintoutMode" maps to the most important choices with the above−mentioned names, and in addition, these names are the same as of the "PrintoutMode" options of other drivers, so the user finds the important printing modes more easily.

The facility of composite options can also be used for other things than for a "PrintoutMode" option, for example a finisher could be controlled by a composite option (to have the most common finishing tasks as "Bound booklet", "Stapled booklet", "Letter in envelope", ...).

Forced Composite Options

Forced composite options are very similar to composite options, but the user cannot set the individual member options, but only the composite option (the user is forced to use the composite option). This allows options acting at two or more places.

Example: A printer driver is a filter which converts a generic bitmap produced by GhostScript to the printer's native format. The command line for converting PostScript to the printer's language could look like this

```
gs -q -dBATCH -dSAFER -dNOPAUSE -sDEVICE=bitcmyk -r600 -sOutputFile=- - | filter -size=<wi
```

where <width> and <height> is the page size in points (1/72 inches). In addition, GhostScript needs to know the page size. For this one usually puts the following PostScript code into the PostScript input file:

```
<</PageSize[<width> <height>]/ImagingBBox null>>setpagedevice
```

where <width> and <height> is again the page size in points. So we need two options for setting the page size, one PostScript option to set the page size for GhostScript and one command line option to set the page size for the filter. The user would have to change both when he wants to print on another paper size, and it does not make sense to have different settings for the two. So one could make the "PageSize" option a composite option of the two, but then the GUI exposes an ugly "PageSize" group with the two individual options. To avoid this, one uses a forced composite option ("Forced" because the user is forced to use the composite option, the individual member options are not accessible).

Assuming that the name of the PostScript option for the page size is "GSPageSize", the name of the page size option for the filter is "filterPageSize" and both have the choices "A4", "Letter", and "Legal", the forced composite option named "PageSize" would look as follows:

```
<option type="enum" id="opt/filter-PageSize">
  <!-- A multilingual <comments> block can appear here, too;
       it should be treated as documentation for the user. -->
  <arg_longname>
   <en>Page Size</en>
  </arg_longname>
  <arg_shortname>
   <en>PageSize</en><!-- backends only know <en> shortnames! -->
  </arg_shortname>
  <arg_execution>
   <arg_order>10</arg_order>
   <arg_section>AnySetup</arg_section>
   <arg_spot>A</arg_spot>
   <arg_forced_composite />
  </arg_execution>
  <constraints>
     <constraint sense="true">
      <driver>filter</driver>
      <arg_defval>ev/filter-PageSize-Letter</arg_defval>
     </constraint>
  </constraints>
  <enum_vals>
   <enum_val id="ev/filter-PageSize-Letter">
    <ev_longname>
     <en>Letter</en>
```

```
    </ev_longname>
    <!-- A multilingual <comments> block can appear here, too;
         it should be treated as documentation for the user. -->
    <ev_shortname>
     <en>Letter</en>
     <!-- Until someone tells me how to learn the user locale in
          backends, the shortname must be monolingual in <en>! -->
    </ev_shortname>
    <ev_driverval>GSPageSize=Letter filterPageSize=Letter</ev_driverval>
   </enum_val>
   <enum_val id="ev/filter-PageSize-Legal">
    <ev_longname>
     <en>Legal</en>
    </ev_longname>
    <!-- A multilingual <comments> block can appear here, too;
         it should be treated as documentation for the user. -->
    <ev_shortname>
     <en>Legal</en>
     <!-- Until someone tells me how to learn the user locale in
          backends, the shortname must be monolingual in <en>! -->
    </ev_shortname>
    <ev_driverval>GSPageSize=Legal filterPageSize=Legal</ev_driverval>
   </enum_val>
   <enum_val id="ev/filter-PageSize-A4">
    <ev_longname>
     <en>A4</en>
    </ev_longname>
    <!-- A multilingual <comments> block can appear here, too;
         it should be treated as documentation for the user. -->
    <ev_shortname>
     <en>A4</en>
     <!-- Until someone tells me how to learn the user locale in
          backends, the shortname must be monolingual in <en>! -->
    </ev_shortname>
    <ev_driverval>GSPageSize=A4 filterPageSize=A4</ev_driverval>
   </enum_val>
  </enum_vals>
</option>
```

This looks exactly like a usual composite option and works also the same way. The only difference is that instead of an "<arg_composite />" tag "<arg_forced_composite />" is used. If the PPD generator finds such an option, it hides the member options by only using "*Foomatic..." keywords to describe them, not any standard PPD keywords as "*OpenUI...", "*OrderDependency...", ... This way PPD–aware graphical frontends do not see the member options but "`foomatic-rip`" has all information from them to run the driver correctly.

String and Password Options

These options allow the user to supply nearly arbitrary strings (within limits of length, characters and structure) to the printer driver, for example names of color calibration files, fax numbers, passwords for confidential jobs, ... Frequently needed strings can be added as enumerated choices, so a frontend can show the option as a combo–box. The enumerated choices are also used for frontends which only support options as defined by the PPD spec. So having enumerated choices is highly recommended for most of these options.

In the XML database string and password options look similar to enumerated choice options. The differences are the option types "string" or "password" and the additional tags to restrict the possible strings.

The "<arg_maxlength>" tags give a length limit, it should once not allow strings longer than around 100 characters, as otherwise foomatic−configure could generate a line longer than the allowed 255 characters in the PPD file when setting the default value, and second, which is very important, it should not allow strings which are too long for the printer filter or driver so that buffer overflows cannot occur. Not using the "<arg_maxlength>" tags makes arbitrary long strings to be accepted, this is not recommended.

With "<arg_allowedchars>" the accepted strings can be restricted to contain only the characters given in the list. This restrictions does not only avoid that the filter chokes on a wrong option, it serves mainly for security reasons, for example to avoid a string like "|| rm −rf * ||" for a command line option. So if the option prototype does not quote the string, command delimiter characters, I/O re−directors, and shell special characters (";", "|", "&", "<", ">", "*", "?", "[", "]", "{", "}", "(", ")", "$", "\", "'", """) should not be allowed. If the string is quoted by the option prototype, the closing quote character and the backslash should not be allowed, so that one cannot escape from the quoting. The allowed characters are checked by a "/^[...]*$/" expression in the Perl scripts, so ranges with "−", a list of forbidden characters with a leading "^", or special characters as "\w", "\d", "\x07", ... are allowed. To allow a backslash, one has to escape it by using two backslashes ("\\").

"<arg_allowedregexp>" allows also to restrict the structure of the string, as it defines an arbitrary Perl regular expression (see "man perlre") which has to be matched by the string. This serves also for having only strings which are usable by the filter and which do not destroy the command line structure. With this one can for example forbid a backslash as the last character to avoid escaping the closing quote of the option prototype. Regular expressions are applied via a '/.../' expression in the Perl scripts. To apply the pattern matching modifiers "i", "m", "s", or "x" (as "/.../i" for case−insensitive matching) begin the regular expression with "(?<modifiers>)" (as "(?i)..." for case−insensitive matching).

It is highly recommended to use at least one of "<arg_allowedchars>" and "<arg_allowedregexp>", as otherwise all characters are allowed in the user−supplied string and so a malicious user can execute arbitrary shell or PostScript commands. If both tags are used, both conditions have to be fulfilled.

Note that for the character lists and regular expressions in the XML files the following character substitutions have to be done:

```
<   -->   &lt;
>   -->   &gt;
"   -->   &quot;
'   -->   &apos;
&   -->   &amp;
```

Here is an example for an option to supply the file name for an ICC profile for the "foo2zjs" driver (this option is neither in the CVS for the Foomatic database nor tested with this driver):

```
<option type="string" id="opt/foo2zjs-ICM">
    <comments>
        <en>
        This option controls which .ICM file to use for color correction.
        ICM files are stored in the directory /usr/share/foo2zjs/icm/.
        </en>
    </comments>
```

```
    <arg_longname> <en>ICM Color Profile</en> </arg_longname>
    <arg_shortname> <en>ICM</en> </arg_shortname>
    <arg_execution>
        <arg_group>Adjustment</arg_group>
        <arg_order>300</arg_order>
        <arg_spot>A</arg_spot>
        <arg_required />
        <arg_substitution />
        <arg_proto>-G%s </arg_proto>
    </arg_execution>
    <arg_maxlength>127</arg_maxlength>
    <arg_allowedchars>A-Za-z0-9\._-/</arg_allowedchars>
    <arg_allowedregexp>(?&lt;!\/)$</arg_allowedregexp>
    <constraints>
        <constraint sense="true">
            <driver>foo2zjs</driver>
            <arg_defval>ev/foo2zjs-ICM-none</arg_defval>
        </constraint>
        <constraint sense="true">
            <make>Minolta</make>
            <model>magicolor 2300 DL</model>
            <driver>foo2zjs</driver>
            <arg_defval>ev/foo2zjs-ICM-DL2312</arg_defval>
        </constraint>
        <constraint sense="true">
            <make>Minolta</make>
            <model>magicolor 2200 DL</model>
            <driver>foo2zjs</driver>
            <arg_defval>ev/foo2zjs-ICM-DL2200RGB</arg_defval>
        </constraint>
    </constraints>
    <enum_vals>
        <enum_val id="ev/foo2zjs-ICM-none">
            <ev_longname> <en>No ICM color correction</en> </ev_longname>
            <ev_shortname> <en>None</en> </ev_shortname>
            <ev_driverval></ev_driverval>
        </enum_val>
        <enum_val id="ev/foo2zjs-ICM-DL2312">
            <ev_longname> <en>File DL2312.icm</en> </ev_longname>
            <ev_shortname> <en>DL2312</en> </ev_shortname>
            <ev_driverval>DL2312.icm</ev_driverval>
            <constraints>
                <constraint sense="false">
                    <make>HP</make> <model>LaserJet 1000</model>
                </constraint>
            </constraints>
        </enum_val>
        <enum_val id="ev/foo2zjs-ICM-DL2324">
            <ev_longname> <en>File DL2324.icm</en> </ev_longname>
            <ev_shortname> <en>DL2324</en> </ev_shortname>
            <ev_driverval>DL2324.icm</ev_driverval>
            <constraints>
                <constraint sense="false">
                    <make>HP</make> <model>LaserJet 1000</model>
                </constraint>
            </constraints>
        </enum_val>

        ...

    </enum_vals>
</option>
```

This option allows to choose either one of the given file names, either by using the "<ev_shortname>"s or the "<ev_driverval>"s, or one can give every arbitrary other file name with a maximum length of 127 characters, only containing letters, digits, periods, underscores, dashes, and slashes, and not having a slash in the end (no directories). Note that in Perl the period must be escaped by a backslash to be taken literally, otherwise it stands for an arbitrary character. The regukar expression for blocking out strings ending with a slash is "(?<!\/)$" (see "man perlre", search for "(?"). Here the slash is quoted by a backslash. In the XML file the "<" is replaced by "<" so that the XML structure does not get broken. "`foomatic-rip`" translates this back before applying the regular expression.

To be able to offer strings as an enumerated choice which are not allowed as an option name in a PPD file, the "<ev_shortname>" may differ from the "<ev_driverval>", the string inserted at the "%s" place holder in the "<arg_proto>" is always the "<ev_driverval>", independent whether the user supplies the "<ev_driverval>" directly or the "<ev_shortname>". In this example both

```
lpr -o ICM= file.ps
```

and

```
lpr -o ICM=None file.ps
```

supply an empty string as the value of the ICM option.

For the default value there must be an enumerated choice, if there is none, the PPD generator will create one. So this entry is allowed (this option is only an example, it is not in the CVS of the Foomatic database):

```
<option type="password" id="opt/Password">
  <!-- A multilingual <comments> block can appear here, too;
       it should be treated as documentation for the user. -->
  <arg_longname>
   <en>Password (for confidential jobs)</en>
  </arg_longname>
  <arg_shortname>
   <en>Password</en><!-- backends only know <en> shortnames! -->
  </arg_shortname>
  <arg_execution>
   <arg_group>General</arg_group>
   <arg_order>100</arg_order>
   <arg_spot>B</arg_spot>
   <arg_substitution />
   <arg_proto> --pass=%s</arg_proto>
  </arg_execution>
  <arg_maxlength>30</arg_maxlength>
  <arg_allowedchars>A-Za-z0-9\.,_\+\=\:-\/</arg_allowedchars>
  <constraints>
     <constraint sense='true'>
      <driver>mydriver</driver>
      <arg_defval></arg_defval>
     </constraint>
  </constraints>
</option>
```

The default value is an empty string here. So the PPD generator will add a choice for the empty string.

Normally, automatically added choices get the same "<ev_shortname>" as the string itself, but if the string is not allowed as an option name in a PPD file, the "<ev_shortname>" will be modified. For an empty string (as in the example above) "None" will be used and all characters except numbers, letters, and underscores ("_") will be replaced by underscores.

The option types "string" and "password" are treated exactly the same way by the PPD generator and by "`foomatic-rip`", the different names are only for frontends to know whether the input field should display the typed characters or asterisks on the screen.

Option Grouping

All options should be put in groups (with the tags "<arg_group>...</arg_group>" in the "<arg_execution>" section of the option XML files, see above). This way many GUIs sort the options into tabs or tree branches according to the groups. This way one gets only the most important options on the first tab and not so often needed ones on additional tabs. This also overrides the automatic option grouping of CUPS (Groups "General" and "Extra").

It is recommended to have the options in groups as follows (plus perhaps special groups, but not one group for every option):

General

Here go options which are most used on a job–by–job basis, as the options for paper type, size, and tray, ink type, duplex, ... and all options affecting the printout quality, as resolution, dithering, ... and especially "PrintoutMode". If a "PrintoutMode" option is present, all quality–related options covered by the "PrintoutMode" option go into the automatically created "PrintoutMode" group (see above). And this is intended, these options are now usually controlled by "PrintoutMode" and so they are not the most important options for the first tab any more.

Do not put color/brightness/gamma, ... options here, they go to "Adjustment".

Options typically to go here are:

- PageSize
- InputSlot
- MediaType
- InkType
- Duplex
- PrintoutMode
- Resolution
- REt
- Dither
- FastRes
- Economode
- ...

All options mentioned after "PrintoutMode" will usually be used as member options for "PrintoutMode", they are only in this group when there is no "PrintoutMode" option.

PrintoutMode

This group only exists if there is a "PrintoutMode" option, because it is generated by this option. It contains the member options of "PrintoutMode". Typical candidates are

- Resolution
- REt
- Dither
- FastRes
- Economode
- ...

They do not need an "<arg_group>PrintoutMode</arg_group>" line, they are put into this group automatically. One should better put an "<arg_group>General</arg_group>" line into these options, so that they go into the "General" group when there is a printer/driver combo for which no "PrintoutMode" option applies.

Adjustment

Options for correcting the appearance of colors, contrast, ..., for head alignment, ... etc. Here most numerical options will go, but also things like "Density", also if it is an enumerated choice option. Typical candidates are:

- Gamma
- Brightness
- Contrast
- Density
- Saturation
- Cyan
- Magenta
- Yellow
- ...

Finishing

If a printer has a stapler, folder, cutter, envelope packer, or similar devices to do additional processing on the ready printout, the options to control this stuff go into this group. Examples:

- Stapling
- Binding
- Cutting
- Booklet
- ...

Miscellaneous

Options which do not fit into the mentioned groups and for which it is not worth to make a special group.

Unprintable margins

On most printers you cannot print arbitrarily close to the borders of the paper. You usually will have margins of certain width on which you cannot print. For filters and application programs to know about these margins PPD files have "*ImageableArea" lines which define the positions of the lower, the upper, the left, and the right borders of the area on which the printer can print. There is one line for each paper size listed in the "*PageSize" option.

To conveniently generating these lines one can use the following XML structure in the Foomatic database entries:

```
<margins>
     <general>
       <!-- The margins here are valid for every paper size for -->
       <!-- which there is no "exception" section -->
       <!-- ---------- -->
       <!-- possible units: -->
       <!-- pt, inches, mm, cm,
       <!-- dotsNNNdpi (NNN: resolution in which dots are counted) -->
       <!-- if "unit" not present, default unit is pt -->
       <!-- ---------- -->
       <!-- if a margin is not present, default width is 0 -->
       <!-- ---------- -->
       <!-- a missing "general" section assumes zero borders as the -->
       <!-- general borders and "pt" as the default unit for -->
       <!-- "exceptions". -->
       <unit>pt</unit>
       <top>9</top>
       <bottom>36</bottom>
       <left>18</left>
       <right>18</right>
     </general>
     <exception PageSize="Photo4x6TearoffTab">
       <!-- if one or more of "unit", "top", "bottom", "left", -->
       <!-- "right" is missing, the appropriate item of the "general" -->
       <!-- section is used -->
       <top>0</top>
       <left>0</left>
       <right>0</right>
     </exception>
     <exception PageSize="A4">
       <!-- It is also possible to give absolute values in PostScript -->
       <!-- coordinates where the origin is the lower left corner. To -->
       <!-- do so, the <absolute /> tag has to be added, otherwise -->
       <!-- the values are the widths of the unprintable margins -->
       <absolute />
       <left>10</left>
       <right>585</right>
     </exception>
     <exception PageSize="...">
       ...
     </exception>
     ...
<margins>
```

This structure is allowed in printer entries in the "<mechanism>" section and in driver entries in the "<execution>" section or inside a "<printer>" entry of the driver's printer list. In the "<execution>" section of a driver entry the margins are valid for all printers used with this driver, in a "<printer>" entry they apply only to the given printer/driver combo.

The shown example could be for the HP PhotoSmart 7150/7350, which does full−bleed only on HP's special photo paper with an 0.5 inch wide tear−off tab on the lower border (and some other paper sizes used in the photo tray). On all other paper sizes the printer leaves white borders of half an inch at the top and at the bottom and a quarter of an inch on the left and right hand side (1 inch are 72 pt). In addition, the page size "A4" allows to print up to 10 points to the left and the right borders.

At first we give the general borders ("<general>" section) where we choose the unit "pt" (PostScript points) for the numbers. These borders are valid for all paper sizes which are not explicitly mentioned with an "<exception ...>" section. For our printers one of the exceptions is the 4x6 photo paper with the tear−off tab (including the tab the paper is 4x6.5 inches large). here the printer prints up to the left, right, and top borders. Therefore we have margins of zero here. At the lower border the printer still leaves half an inch white (therefore probably HP introduced the tear−off tab), so we keep the 36 pt of the "<general>" section by not mentioning a new lower border. For A4 we redefine the left and the right border. This is also possible in absolute PostScript coordinates measured from the lower left corner, as we do here. We indicate this with the "<absolute />" tag. The left border is at 10 pt from the left, and as A4 paper is 595 pt wide, the right border is at 585 points from the left.

One hint for the choice of the units: Float numbers as border widths are allowed, but it is recommended for having exact info to choose a unit which gives integer numbers for the widths (which is always possible with the "dotsNNNdpi" unit with NNN being the maximum resolution of the printer).

A "<margins>" section in a printer entry should represent the printer's hardware capabilities. Such a section in a driver entry should represent how the driver's limitations are. If there are margins defined in both the printer and the driver entry of the desired printer/driver combo, the more restrictive (wider) borders count. If there are no border definitions in both the printer and the driver entry, the borders are assumed to be of zero width (full−bleed).

Adding arbitrary extra entries to the PPD file

The "<ppdentry>" tags allow to add extra lines to the PPD file. The tags can be put into the top level ("<printer>") of a printer XML entry, into the "<execution>" section of a driver XML entry, or into the "<printer>" entries of the printer list in a driver XML file. They serve mainly to put a default resolution into PPD files for drivers without "Resolution" option. Examples:

"hpijs" driver, default resolution for HP DeskJet 350. For this driver the default resolution depends on the printer class. Therefore the appropriate "<ppdentry>"s have to be in the printer entries of the printer list:

```
<driver id="driver/hpijs">
  <name>hpijs</name>
  ...
  <printers>
   <printer>
    <id>printer/HP-DeskJet_350C</id><!-- HP DeskJet 350C -->
    <ppdentry>
      *DefaultResolution: 600dpi
    </ppdentry>
    <margins>
     <general>
      <unit>in</unit>
      <relative />
```


```
      <left>0.25</left>
      <right>0.25</right>
      <top>0.125</top>
      <bottom>0.67</bottom>
     </general>
     <exception PageSize="A4">
      <left>0.135</left>
      <right>0.135</right>
     </exception>
    </margins>
   </printer>
   ...
  </printers>
</driver>
```

"pnm2ppa" driver: This driver has no "Resolution" option, and all printers print in 600 dpi with it. So we put the "<ppdentry>" into the "<execution>" section:

```
<driver id="driver/pnm2ppa">
  <name>pnm2ppa</name>
  <url>http://sourceforge.net/projects/pnm2ppa/</url>
  <execution>
   <filter />
   <prototype>gs -q -dNOPAUSE -dPARANOIDSAFER -dBATCH -r600%A%Z
-sOutputFile=- - | pnm2ppa%C%B -i - -o -</prototype>
   <ppdentry>
    *DefaultResolution: 600dpi
   </ppdentry>
  </execution>
  ...
</driver>
```

Note that leading spaces are removed from the lines between the "<ppdentry>" tags before they get inserted into the PPD file.

The lines are added at the end of the PPD file header, right after the lines for the basic hardware capabilities of the printer.

What is planned for the future

Here are some ideas which we want to implement in the next releases of Foomatic. Main focus is the maintainability of the Foomatic database. More ideas and discussion you can find on:

- The Foomatic Developers forum/newsgroup/mailing list: http://www.linuxprinting.org/newsportal/thread.php3?name=linuxprinting.foomatic.devel or http://www.linuxprinting.org/cgi–bin/mailman/listinfo/foomatic–devel, archives http://www.linuxprinting.org/pipermail/foomatic–devel/
- The TODO list in the "Progamming" section of the "Contributing page": http://www.linuxprinting.org/contribute.html#programming
- Some ideas and sketches of implementation: http://www.linuxprinting.org/Foomatic–Devel–Ideas.txt

Web interface for adding new printer entries on linuxprinting.org

Most of the printers listed on linuxprinting.org were added by users, using a web interface. With

the switchover to an XML database this web interface disappeared and so an important source for printer entries. As keeping the printer database up−to−date is a lot of work, the best is to revive the web interface.

To avoid all the noise (as duplicate or empty entries, entries with wrong information) we will let the user's contributions go into a queue and make an administrator interface to check, correct, and finally submit the entries into the database. We could make it possible for driver authors and printer manufacturers to directly publish the entered info.

Printer/Driver classes

One problem of the Foomatic database is, that one has often to repeat the same comments in many printer entries or that one has to add constraints for many individual printers into the option entries or into the printer lists of driver entries. This makes maintaining the database a lot of work.

To solve this problem we came to the idea of introducing printer classes. A class should contain printers with a common capability, as for example PCL−5e printers, wide format printers, printers compatible to the HP DeskJet 990C, ... So one can simply put the PCL−5e printer class as the only printer entry into the printer list of the "ljet4" driver and simply add any new PCL−5e printer to the PCL−5e class. Then the printer gets automatically associated with the "ljet4" driver. In a "PageSize" option one could make all paper sizes bigger than Legal only available to the wide format class. There could also be a class of all multi−function devices needing HPOJ to be able to scan, this class would add an appropriate text paragraph to the printer's web page on linuxprinting.org. And when this text piece needs to be updated, one updates it in the class entry and it automatically changes on hundreds of printer pages.

This makes it also easier for users to add a printer via a web interface, They simply need to pick classes to which the printer belongs and so most information and driver/option associations are done, the user only needs to add little info by filling in the web form, the maintainers of linuxprinting.org need only to do some fine tuning.

Similarly one could also make classes for drivers.

Option conflicts

An important structural element of PPD files which is not supported yet by Foomatic are option conflicts. Option conflicts are definitions in the PPD files which prevent the user from making choices which make printing impossible or simply do not make sense, as for example duplex on transparencies or A3 paper in the envelope feeder. This avoids that users send print jobs with bad settings and then complain about something wrong or nothing coming out of the printer. This is especially important because most spoolers cannot report back error messages of the driver to the user who has sent the job. So users do not know why nothing comes out of the printer. Or they waste expensive material when nice double−sided transparencies come out.