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

(IV.) Foomatic from the Developer's View:

How does Foomatic work?

Table of Contents

(IV.) Foomatic from the Developer's View: How does it work?

(IV.) Foomatic from the Developer's View: How does Foomatic work?

Presented by Till Kamppeter, maintainer of linuxprinting.org, leader of the Foomatic project, author of XPP, and responsible for the printing part of Mandrake Linux

What we will show: Why Foomatic? – History of linuxprinting.org – Database structure – How are configuration files generated – Data flow when printing with Foomatic – What are the plans for the further development?

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 and pas 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 on 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 around 900 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 driver configuration files or even complete print queues for all known free spoolers (CUPS, LPD, LPRng, GNUlpr, PPR, PDQ, and spooler–less printing). Foomatic–based print queues allow the user to adjust all options of the printer driver on a per–job basis. And 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 the original "PPDs".

How CUPS uses the original PostScript PPD files

The CUPS server associated the appropriate PPD to each queue. It could parse the PPD and extract the user–available print options for the related target printer. Clients would 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 could translate the list of printer options into nice dialogs. The client just selects the target printer and sends the desired print options as commandline parameters to the CUPS server, along with the printfile (in most cases PostScript). The CUPS

server would then, based on the contents of the PPD for the individual options, insert the right command 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 print choices 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 didn't limit its PPD support to PostScript printers and their RIPs only. It rather extended 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 print options (and related low−level printer commands) available for some popular inkjets, deskjets, laserjets and dot matrix printers into a PPD−conforming syntax and could thusly handle those printers inside the same framework as the real−PostScript models. CUPS added one simple line to the PPD ("*cupsFilter..."), which told the CUPS server how to handle the printfile. 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 created a brilliant concept for handling print options for all printer types through PPDs −− however it didn't deliver many PPDs. It shipped a few sample PPD files, which were generic enough to support a few hundred LaserJet, DeskJet, Stylus Color, Stylus Photo 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 printfiles 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, 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. His database contained most of the complicated, differing Ghostscript commandline parameters, which made different Ghostscript filters (="devices") and models work. This info could surely go into a PPD−like file, too, describing a driver setup for a Ghostscriptfilter/printermodel 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 release of "CUPS−O−Matic" appeared. It was an online−working generator of PPD−files for the use with CUPS. You could use a browser, surf to the CUPS−O−Matic page, select your model, get an appropriate driver (or Ghostscript device)

suggested, click "OK" and generate on the fly a PPD file. It would need an additional "cupsomatic" Perlscript installed as a faked CUPS filter, which would then handle the printfile instead of CUPS.

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 is not fit for handling medium to large sized network printing environments).

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 certain 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.

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 giving at "LinuxTag 2002" in Karlsruhe in Germany. They joined the Foomatic developer team and a new design for Foomatic where all spoolers will use PPD files as printer/driver description files is 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 is participating in the development of Foomatic 3.0, so that will also use Foomatic in the future. In addition, around 5000 people every day visit the linuxprinting.org web site.

The current Foomatic – the 2.0.x series

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 lets see the "ljet4" driver, a driver for printers which understand PCL 5 and with maximum resolutions up to 600 dpi. Its printer list contains all printers which understand PCL 5: 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 5 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 5 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.

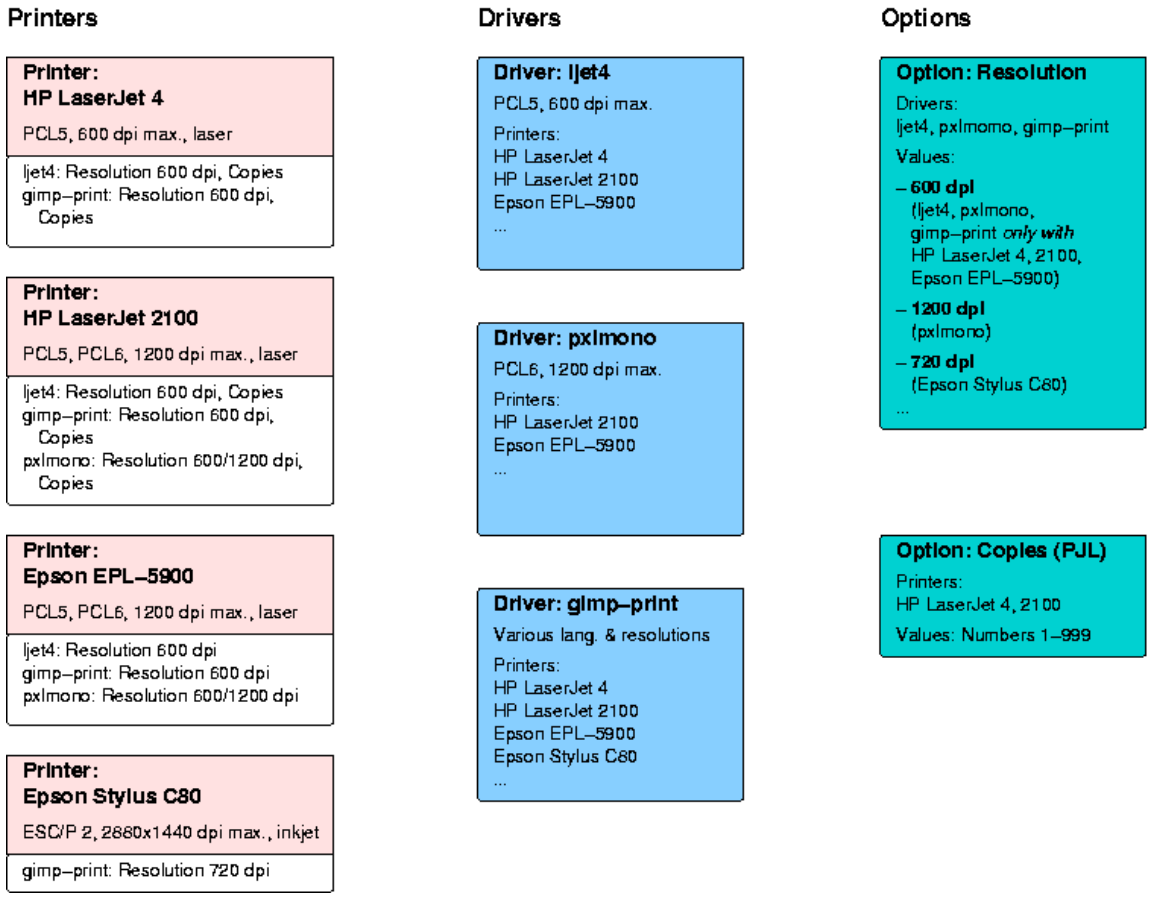

Fig. 1: An example to explain the structure of the database

The second option in our example is the numerical option "Copies", it allows integer numbers from 1 to 999 as settings, 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 configuration file for the desired printer/driver/spooler combo. The principle of obtaining this is the same for both generating it on the linuxprinting.org 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. It is inserted into all the spooler-specific configuration files so that the filter scripts (which are written in Perl) know about how to use the driver and how to apply the user-supplied option settings.

Now the spooler-specific configuration file is generated by wrapping spooler-specific stuff around the Perl data-structure and, if necessary, hiding the Perl code in comment lines so that the spooler does not choke on it. This 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 configuration file a filter script has to be installed which will be called by the spooler to translate the incoming PostScript job data into the printer's native language. Depending on the spooler the following filters are used

CUPS:	`cupsomatic`
LPD. LPRng, GNUlpr:	`lpdomatic`
PPR:	`ppromatic`
PDQ:	Filter code integrated in the configuration file
No spooler:	`directomatic`

The filters are Perl scripts and read at first the printer and driver information from the Perl data structure in the configuration file. For that no extra Perl libraries or C programs are needed. The filters get 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 filter 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 configuration file, a filter, and, if the configuration file is not already a PPD file (as with CUPS or PPR as the spooler) an optional PPD file 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 a 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 (brown lines from the right) and CUPS itself takes the printer option information from the PPD file of the appropriate print queue. The PPD file is the configuration file generated by Foomatic when setting up the queue. In case of LPD, LPRng, or GNUlpr being the spooler "`kprinter`" reads the Foomatic configuration file directly (brown lines from the left) and "`gpr`" reads the Foomatic–generated PPD file. "`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).

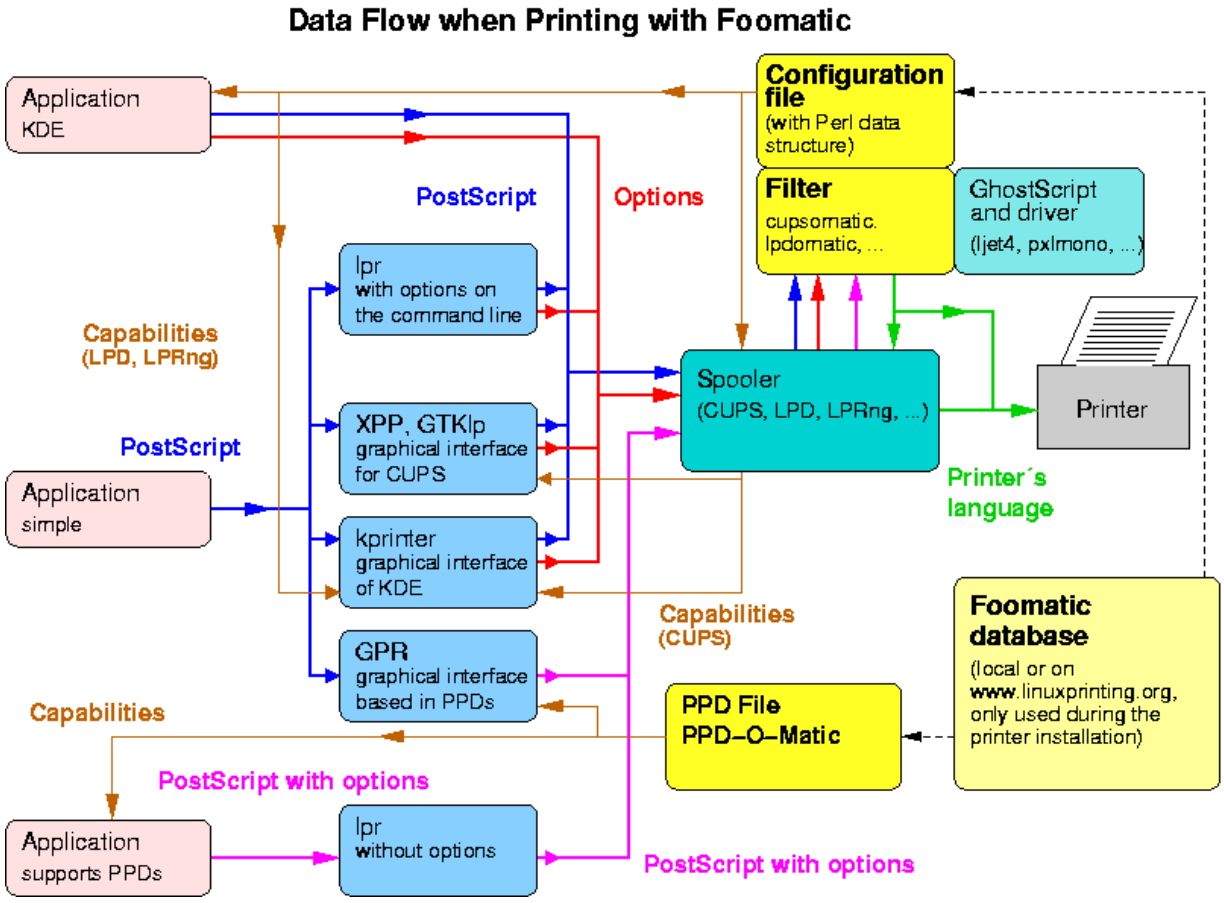

Fig. 2: Data flow when printing with Foomatic

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 the appropriate Foomatic filter script and hands over all the PostScript and option settings data to it. Then the filter builds the GhostScript command line according to the users 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 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" or "float" 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 user's value gets put into the arg_proto's %s location.

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 in the beginning of the PostScript data stream of the job, not after the code for the first page begins. The <arg_proto> lines of <arg_pjl> are PJL commands which are sent to the printer before the output of the drivers 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 marked with <nopjl /> are for example "hpijs" and "hl1250".

```
<arg_execution>
 <arg_order>100</arg_order>
 <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>
```

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 "/".

A printer entry: HP LaserJet 4000 (db/source/printer/100576.xml)

The printer file contains information specific to a particular printer.

```
<printer id="printer/100576">
```

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>
```

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</u
```

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. A complete entry could look like:

```
  <autodetect>
    <parallel>
      <commandset>MLC,PCL,PML</commandset>
      <description>Hewlett-Packard DeskJet 660C</description>
      <manufacturer>HEWLETT-PACKARD</manufacturer>
      <model>DESKJET 660C</model>
    </parallel>
  </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;
# 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;
```

```
  <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 &quot;Fastres 1200&quot; in the printer configuration
    options.&lt;p&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>
 <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/240137</id><!-- Alps MD-1000 -->
  </printer>
  <printer>
   <id>printer/240169</id><!-- Alps MD-1300 -->
  </printer>
  <printer>
   <id>printer/240105</id><!-- Alps MD-2000 -->
  </printer>
  <printer>
   <id>printer/240073</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/62304</id><!-- HP LaserJet 4050 -->
   <comments>
    <en>to 1200dpi</en>
   </comments>
  </printer>
```


What is planned for the future – the 2.9.x series

Here are some ideas which we want to implement in the next generation of Foomatic. 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 and http://www.linuxprinting.org/cgi−bin/mailman/listinfo/foomatic−devel and http://www.linuxprinting.org/pipermail/foomatic−devel/2002q3/thread.html
- 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

Adobe−compliant PPD files for all spoolers

The main change will be that the configuration files for all spoolers will be PPD files. All differences between spoolers will be taken care of in the filter scripts. The PPD files will also not contain the Perl data structure any more, all information is provided in the Adobe−compliant PPD format.

Advantages:

- There is only one type of configuration files: PPDs
- For PPD−aware applications one uses the PPD which also serves as the configuration file for the spooler, independent what spooler is used
- Manufacturer−provided PPDs can be used with the same filters as PPDs provided by Foomatic, independent whether they are for PostScript or non−PostScript printers or whether they use Foomatic features or not. So all spoolers will get fully PPD−aware and printer manufacturers can easily provide drivers which work in every printing environment.

Collective options

This is a new 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.

For example we could have a "Document Type" option with the following choices:

- Draft
- Office document
- Photo

It would set the individual options "Media Type", "Resolution", and "Dither" as follows:

```
Choice           Media Type      Resolution        Dither
-------------------------------------------------------------------
Draft            Plain Paper     300x300 dpi       Fast
Office document  Plain Paper     600x600 dpi       Floyd-Steinberg
```

```
Photo           Photo Paper     1200x1200 dpi     Floyd-Steinberg
```

The individual options will have a setting named "Default" which will be the default setting and when "Default" is chosen, this option is set by the collective option. When the user has made a "real" choice in the individual option, this overrides the appropriate setting of the collective option.

Options not being member of a collective option (as "Page Size" and "Media Source") are always set individually by the user.

Features of Adobe's PPD format

To present the options in a more intuitive and ergonomic way and to prevent the user from getting unexpected results we want to add two important features of the PPD standard:

- *Option groups*: Options can be put together in groups to sort them by their logical context. One could have groups for quality settings, paper settings, finishing, etc. One could also put the member options of a collective option into a group, so that they stand on an own tab and the collective option stands on the main tab to give newbie−friendly, not overloaded option dialog. With sub−group one can even order the options in a tree structure.
- *Option constraints*: Option constraints (or conflicts) prevent the user from making choices which make printing impossible or simply do not make sense (as Duplex on transparencies or 1200 dpi on plain paper).

Splitting Foomatic in various packages

As in the new PPD−centric Foomatic the filters will also be able to use non−Foomatic PPDs they should be separated from the main Foomatic packages with the database engine and the printer configuration scripts. Also the printer database should be a separate package so that one can update it without needing to update the software, too.

The packages could look like this:

- *foomatic−db−engine*: The software reading the XML database and generating PPD files from it and also generating print queues: DB.pm, foomatic−configure, ... versioning: 2.9.x
- *foomatic−filters*: The software needed for queues which Foomatic PPD files to work: foomatic−rip, cupsomatic, lpdomatic, ppromatic, directomatic, pdqomatic, ... foomatic−filters can be used without the other packages, when one has pre−compiled PPDs, or when one has manufacturer−supplied PPDs of native PostScript printers, so this makes every PPD working with every spooler, versioning: 2.9.x
- *foomatic−database*: The XML database with all printer entries and all manually generated driver and option entries, versioning: date in the format yyyymmdd.r as for the first release of today 20020716.1. By automatic mechanisms every day a tarball is generated and probably an extra tarball with every CVS tag (for driver/distro releases).
- *foomatic−database−hpijs*: The XML data for the HPIJS driver (only driver and option entries). The Makefile will build the entries with the hpijs data generator, versioning: <HPIJS version>.<release>, as the second release for HPIJS 1.1 will get 1.1.2.