dvilj

A DVI printer driver for *HP Laserjet* printers

By Gustaf Neumann, edited by Karl Berry.
RISC OS port by Andreas Dehmel.

February 28, 1996

This is the RISC OS port of Gustaf Neumann's `dvi2xx`, with changes by Karl Berry and some very useful additional bits by me (Andreas Dehmel). The entire release consisted of a `tar` archive about 2 MB in size. I won't include everything in this port, as most of that space was taken up by fonts. You can get the entire package from `ftp.dante.de` or any other CTAN ftp server[1]. Most people won't be needing those particular fonts anyway, since they're only .tfm and .vf files to enable you to use the printer resident fonts from TEX; if you only want to use standard TEX fonts you won't be needing any of these.
What I do provide is a directory scan of the original release from which I started porting this program. This and some of the `readme` files are included in the directory `orig_docs`.

1 What is it?

dvilj is a DVI printer driver for *HP Laserjet* printers. It works by downloading fonts to the printer and printing the DVI file as text. What I've added is support for diagrams as used in DVIview version 1.00. That means that diagrams are printed as bitmaps and the resulting PCL raster data is inserted into the output file. The binary included in this release, which I will refer to as **dvilj** from now on, just for convenience, was compiled for the LJ4/5 series; that means it supports 600 dpi printing which is the default resolution! If you need a version for an older LJ model you'll have to recompile the source with the corresponding makros defined. More about this in section 8.

1.1 Why use dvilj instead of just printing from DVIview?

That's easy to answer: speed and printout size. For example the entire documentation for **dvips** (49 pages) takes under 40 seconds to print on my A5000, the printout size is under 1MB and yes that's at 600 dpi. Who needs PostScript?
Usually drivers like this have the disadvantage that they don't support platform-dependent stuff such as diagrams. But I added support for that as well so now you get the best of both worlds. The only disadvantage is you'll have to use it from the command line.

[1] Comprehensive TEX Archive Network; other sites include `ftp.shsu.edu` and `ftp.tex.ac.uk`.

1.2 Any drawbacks?

Well... you'll be needing lots of .pk or .pxl files[2] on your harddisc which take up a couple of MB and some time to create[3]. You'll also have to get to grips with METAFONT to a certain extent in order to be able to create those fonts. So if all you do is print the odd DVI page every now and then you're probably better off printing everything as graphics the way the standard printer drivers do. If you're printing more than that you should definitely consider using this driver. If you're printing a lot there's practically no alternative ;-).

2 Installation

First up I'm assuming you have a working TEX version already installed on your harddisk. Unless that is the case there's little use in installing **dvilj** on your computer.
Simply copy the executable `dvilj` into your `!TeX.bin` directory. Furthermore **dvilj** needs the system variable `TexFonts$Path` to be set up and pointing to those directories that contain the .mf, .tfm and .pk or .pxl files. **dvilj** itself only needs .tfm and .pk or .pxl files but in case some of the bitmap fonts are missing[4] you'll have to create them using METAFONT and **dvilj** will attempt to help you with that for which it needs to find the .mf files as well (see section 3.3).
The newer TEX releases have `TexFonts$Path` set up correctly by default, in case you've got an older release I'm afraid you'll have to fiddle with it a bit. Make sure you read section 4.

3 Usage

I won't explain all the switches here; look in the directory `orig_docs` for a short introduction on what everything does. I'll merely deal with the stuff I've added.

3.1 Diagrams

As I mentioned, **dvilj** supports diagram printing, i.e. draw files or sprites inserted in a DVI file with the `drawfile` or `DVIview_diagram` `\special` command will be printed as graphics which are then appended to the output file created so far. This means

1. If you want to print diagrams you'll have to have a Laserjet printer driver loaded.
2. You can't print diagrams from a task window! You have to go single tasking for that.
3. If you want to print draw files you'll have to have the drawfile module either loaded or in your `system:modules` folder. **dvilj** will attempt to load it from there in case it's not already present.
   Note: the drawfile module is *not* part of this release; these days it seems to be part of

[2] packed bitmap fonts
[3] usually around 1-2 minutes per font
[4] this will usually be the case when you're using a driver like this for the first time

pretty much anything and everybody who has the new version of DVIview will have it anyway. In case you really don't have a look at `ftp.acorn.co.uk`.

However **dvilj** offers you a way to cache those printouts so ideally you'll have to actually print diagrams only once and can on future runs simply include the pre-printed files. This also means that you won't be needing the drawfile module and you can print from a taskwindow.

3.1.1 The Diagram Cache

Cacheing diagram printouts works like this: **dvilj** determines the diagram's leaf name and saves the printout of that diagram with that leaf name in the cache directory (see below).
This means that if you want to cache your diagrams you'll have to have *unique leaf names* for all your diagrams. The alternative would have been to store the printouts in the same directory as the original diagram and hacking the name a bit. I can't think of a decent system to name these printouts, though, and most of all I don't like the idea of having rather large files all over my harddisk; so I prefer the cache directory approach by far.
The *cache directory* is created in the document folder. It consists of a base name 6 characters in length[5] with a decimal number appended. That number is the base resolution in dpi (e.g. 600) times the current magnification. This system ensures that **dvilj** doesn't use printouts that were created with the wrong dimensions when you're printing at a different magnification or resolution. On the other hand if you're printing at 300 dpi with a magnification of 1 you'll be getting printouts with exactly the same dimensions as if you had been printing at 600 dpi with a magnification of 0.5; that's the reason for using the product of these two parameters.
The default base name for the cache directory is `LJdiag`, so diagrams printed at a base resolution of 600 dpi with magnification 1 will be stored / looked for in `LJdiag600`. Changing magnification to $\frac{2}{3}$rds will result in a cache directory of `LJdiag400` and so forth.
Whenever **dvilj** encounters a diagram it first attempts to find a file with the same leaf name in the appropriate cache directory . If such a file is found and its time stamp is from a later date than that of the actual diagram then the printout is used directly. Otherwise a printout is created using the currently active printer driver; this printout will be either `<Wimp$Scrap>` (no cacheing, gets deleted after use) or the name of the cache directory plus the diagram's leaf name (cacheing). This system may lead to some inconsistencies which will be discussed now.

3.1.2 Keeping the Cache Directory consistent

First up I'd like to explain when you'll be needing a new printout of a diagram and when you won't. Forcing **dvilj** to make a new printout is simply a matter of deleting the obsolete printout file by hand. Determining when to do this is basically very simple:

- You need a new printout when you change the diagram's magnification (using the `scale` command) or change its cropping values (using the `crop` command). Likewise a new printout is needed if the diagram itself has changed, of course.

[5] and is truncated to that length in case it's longer

- You don't need a new printout if the diagram has merely changed position. It's one of the truly wonderful aspects of PCL that you can place raster graphics anywhere on the page.

dvilj can't know if you've changed a diagram's `scale` or `crop` values. If you've done either of these you should delete the diagram's printout from the cache manually to make sure a new printout is created the next time you're printing your document.
Although **dvilj** compares time stamps this only reliably detects if you've updated a diagram. Copying files usually means that the copy inherits the time stamp of the original, so in case you've replaced a diagram with another you should delete the diagram's printout by hand as well.
If you remember those points you shouldn't have any problems with this cacheing mechanism.

3.1.3 Switches related to diagram cacheing

Diagram printing has added three switches to those already present. These are

`-i<filename>` : Change the cache directory's base name to `<filename>`. This should *really* not include a ".."!

`-j` : Don't print diagrams. This means **dvilj** just leaves blank space where the diagrams are supposed to be. Good for preview printouts as it keeps the size of the printout down.

`-k` : Cache diagrams. By default this is off. Even when it's off **dvilj** will try to get printouts from the cache directory (it just won't store any there). If you don't want to use printouts present in your cache directory/ies but don't want to delete them either you could change its base name with the `-i` switch.

One word of advise concerning the DVI file name you pass to **dvilj**: give it the *full* name, including the `.dvi` postfix. Otherwise things like the cache directory might be created in the wrong directory.
When printing, **dvilj** will echo `Diagram: Printing <`*diagram*`>(`*file type*`)`. If the file is to be cached, `(C)` will be appended. When the resulting printout is appended, its filename will be echoed in square brackets.

3.2 Printer driver requirements

The printer drivers only come into it when you're printing diagrams, of course. I have no idea if the output the Turbo Drivers produce is usable with **dvilj** though. The crucial point is how I'm scanning the printout. Obviously I have to strip away certain parts of the printout to avoid PCL commands related to printer resetting, paper format, positioning and page feed. Basically what I'm doing is include everything from `ESC,*,r,...` (raster graphics on, inclusive) to the final `0x0c` character (page feed, exclusive). I've tested my approach with a lot of different printouts with the LJ2, LJ4 and LJ5 printer drivers included in Printers 1.52 and they all conform nicely with this. However whether the Turbo Drivers comply as well I really don't know.
dvilj assumes that the LJ driver you're using *creates raster data only*, no font downloading

and things like that! This is the case with the Acorn printer drivers; in fact this is the reason why **dvilj** is far superior to standard printer drivers as far as printing DVI files is concerned. Should there ever be a LJ printer driver that downloads fonts you're bound to get problems using it with **dvilj**. However in that case **dvilj** would be rather superfluous.

3.3 Missing fonts

In case **dvilj** is missing bitmap fonts it will give out a warning, naming the missing font. There are a number of ways for a program to help the user create these fonts; `dvips` for instance creates the fonts itself on the fly. On the other hand **dvilj** needs a couple of hundred kB to begin with and METAFONT needs almost 1 MB so I have my doubts whether it would be wise to demand a wimpslot of around 1.5 MB as this would necessitate. Not to mention the fact that `dvips` has crashed the entire desktop on me on a number of occasions while creating fonts.
Therefore I've opted for creating a task obey file that contains all the necessary commands to METAFONT and related programs to create the fonts. This file is created in your document folder under the name `lj-fonts`. Simply double-click on it after **dvilj** has terminated[6] and all the missing fonts should be created in a taskwindow. Since you'll be needing modes for METAFONT named `hplaser`[7] and `ljiv`[8] you should read section 4 first, though. If you whish to use other modes you'll have to edit the task obey file by hand.
In case you're missing the .mf file for a particular font **dvilj** will warn you about that. This means that no bitmaps can be created for this font and thus nothing will be added to `lj-fonts`. In a case like this I recommend looking up the missing fonts at a CTAN server and kiss some more of your harddisk space goodbye.

4 Metafont issues

If you want to run **dvilj** at 600 dpi you need a METAFONT mode for that resolution. The standard TEX release doesn't have such a mode defined, so you'll have to create one first.
Look in your `!TeX.Formats` directory for the file `local.mf`. This includes definitions for all the modes METAFONT understands. Add a definition for a 600 dpi mode; I recommend naming it `ljiv`: this is the name used in the `dvips` distribution and I've adopted the same name for **dvilj**. The definition should look something like this:

```
mode_def ljiv =
   proofing:=0;
   fontmaking:=1;
   tracingtitles:=0;
   pixels_per_inch:=600;
   blacker:=0.5;
   fillin:=.2;
```

[6] because it will be open while **dvilj** is running
[7] 300 dpi, included in the standard TEX release
[8] 600 dpi, usually has to be created first

```
o_correction:=.6;
enddef;
```

I don't know if the reason are 600 dpi but with a `blacker` value of 0 the fonts look *extremely* thin on my HP LJ 5L, especially with small characters. So I recommend a `blacker` value of at least 0.5, although I'm pretty sure the METAFONT wizards will want to see my head on a spike for that. And while you're at it change the `blacker` value of the `hplaser` printer definition to 0.5 as well.
Next you'll have to recompile your METAFONT base file. You'll need a rather big wimpslot for this (around 1280k should do). Now change your current directory to `!TeX.Formats` and type

```
wimpslot 1280k
inimf &plain \input local.mf; dump
```

This will create the files `base` and `log` in your `!TeX.Formats.local` directory. Delete `log` and copy `base` in your directory `!TeX.Formats.plain`, overwriting the `base` file already present there. From now on you'll be able to use the 600 dpi mode by passing `mode=ljiv` to METAFONT (which is usually called `virmf`).

For more information on METAFONT I recommend the crash course, *Metafont for beginners*, which can be obtained from any CTAN server.

4.1 Errors running METAFONT

The task obey file `lj-fonts` sets up a wimpslot of 960k which is enough for METAFONT so you shouldn't be getting an error about the wimpslot being too small. One error you might run across is METAFONT complaining that it can't write the .tfm file for the font just created. This can be the case when the .tfm file already present

- is locked or
- was created with a different mode / resolution.

In both cases the best option IMHO is to delete the old .tfm file. Again, I feel someone sharpening the spike...

5 Thumbnail printing

Yes, you can also print thumbnails with **dvilj**. This is where **dvidvi** comes into it which is the reason why I've included it in this release.
dvidvi is a useful little program that converts DVI files into other DVI files. You can select a range of pages using various switches[9] and you can also place multiple pages of the input DVI file on one page of the output DVI file using the `-m` switch. This is followed by the number `p` of pages to be placed on one page. Next comes a colon and a comma separated list of the

[9] for more information refer to the `readme` file in the **dvidvi** directory

format `n(offx,offy)` for all the pages in question. `off,offy` are the offsets[10] at which to place the top left edge of the page. Note that with DVI files x increases from left to right but y increases *from top to bottom*! The number `n` is the number of the source page modulo `p`.
What **dvidvi** can't do is change magnification, rotate or change paper format. That's what your printer driver has to perform and - luckily - **dvilj** can do that. Now for some examples:

5.1 2 pages per page, landscape

In this case you'll call **dvidvi** with

```
dvidvi -m2:0(0,0),1(8,0) infile.dvi outfile.dvi
```

i.e. place the first page at the origin and the second one moved 8 inches to the right. That's assuming that you're using A4 format which is around 8×12". Next you'll call **dvilj** with

```
dvilj -l -m#667 outfile.dvi
```

i.e. select landscape mode and set magnification to $\frac{2}{3}$rds[11]. I don't think all Laserjets are supporting landscape mode, though. LJ4 and LJ5 models definitely are.

5.2 4 pages per page, portrait

This will require the call

```
dvidvi -m4:0(0,0),1(7,0),2(0,10.5),3(7,10.5) infile.dvi outfile.dvi
```

to **dvidvi**. The offsets are a little different this time but experimentation showed that these are best for this format, at least on the LJ5 series. Now the call to **dvilj** will have to be

```
dvilj -m#500 outfile.dvi
```

Be warned that changing the magnification means you'll need different bitmap fonts. And you thought you could give METAFONT a rest for a while...
A word of advise on thumbnail printing: you should really use this only if your printer can deal with 600 dpi or the document has rather large print. The deficiencies of 300 dpi printing will become painfully obvious the smaller your fonts are getting.

6 Legal issues

Gustaf Neumann's `dvi2xx` is public domain for all I (and Karl) know, Karl Berry's modifications are covered by the general GNU software license. My parts in this project (mainly the diagram printing and parsing) are public domain as well. You may copy this program and its documentation freely as long as you're not trying to sell it for profit. You may also change the source code although I'd prefer it if you sent me a mail about what it is you don't like so

[10] by default in inches, but you can use other units as well
[11] a magnification of 1 is achieved by `m#1000`

I can try fixing it. You may *not* modify this documentation! If you wish to make annotations write them in a different file.
This program comes with *NO WARRANTY*! I will not accept any responsibility for any damage caused by the use of this program. This doesn't mean it's full of bugs, though. On the contrary it's been working wonderfully for me.

7 Contacting me

Here's my address in case you wish to send me large amounts of money, donate hardware (I really could use a bigger harddisk) or something much less improbable such as bug reports. Please only send bug reports related to the diagram printing part to me as I haven't written the font printing engine. Contact Gustaf or Karl for this. Here we go:

Andreas Dehmel
Am Schorn 18
82327 Tutzing
Germany

email: `dehmel@informatik.tu-muenchen.de`

The remaining part of this documentation is of interest for programmers only.

Figure 1: And here at last we have a figure...

8 For Programmers

So you think you're tough, huh ;-) ? OK, let me try to explain some things you might need to know. First up I've been using GCC for this port, the object files were linked to the SharedCLibrary using `gstubs`. I checked for the compile platform by testing for the makro `__riscos`. GCC has this defined; if your compiler doesn't nothing will work...

8.1 A short introduction

I'd like to start this section by telling you a thing or two about the program how I got it. The sources I got were

`dvi2xx.c` - The main source code.

`tfm.c` - Reading information from .tfm files.

`commands.h` - Some makros defining DVI command codes.

`config.h` - Some basic configurations.

What was missing, unfortunately, was `findfile.c` and the `kpathsea` functions. Well, `findfile.c` was easily written after a close look at the source; it also contains some additional functions. `kpathsea` is not necessary so I left it out.
`dvi2xx.c` was *one* big 150k file of C code with more preprocessor switches than you can shake your finger at. The first thing I did was split it up into a number of smaller files with functions grouped by functionality as well as moving all the declarations and prototypes into a separate header file, `dvilj.h`. There still might be some inconsistencies here but it's a lot better than it used to be and I don't have to wait 2 minutes for the compiling to end when all I did was change one line of code.
One of the first things I noticed was that I had to replace all `putc` and `getc` calls with `fputc` and `fgetc` as otherwise I got serious problems when linking to the SharedCLibrary. I don't know if that's a problem with the SharedCLibrary or a side effect of using the UnixLib headers. Probably the latter. The next thing I noticed was *very* strange behaviour, as the program would run for about 17 pages and then suddenly break for no apparent reason. No matter what I tried I couldn't fix it so I decided to direct all reading and writing operations directly to the OS, using calls to OSlib. This turned out to be a major task and took up some time and lots of modifications. For instance there was

```
#define EMIT  fprintf
```

which had to be changed somewhat. I printed into a string using `sprintf` and wrote that out later. This necessitated two more makros, the destination of the `s/fprintf` command, `EMTO`, and the writing part, `EMFLUSH`. In case you're using the standard ANSI C functions this will result in exactly the same code as before :-). I also tried to make the whole output more consistent: *everything* going into the printout file will now use a function starting with `EMIT`. After I finally got it to work I found out that now all of a sudden it worked just as well using the standard ANSI C IO functions. I must have eliminated the root of the problem in passing when I was rearranging the program's IO. So you may compile the program using either

ANSI C functions or system functions. For the latter case define the makro RISC_USE_OSL, otherwise the ANSI functions will be used. The binary included in this release was compiled with the ANSI functions.
The next thing was trying to get it to run faster by setting up buffers. The result was not exactly overwhelming so I'm not using them any more. But you can enable them by defining the makro RISC_BUFFER. One instance where I *am* using an internal buffer is when appending printout files of diagrams to the actual printout, so you may not remove the input buffer from the program. I've already made sure nothing is declared that doesn't have to be.

8.2 Diagrams

After everything was working alright I shelved the whole thing for a couple of weeks. Then I started working on the diagram printing mechanism.
This is rather straightforward and the entire printing code is included in the file diagrams.c. It contains the functions

AppendRasterFile(char *rasterfile): appends the raster data contained in the file rasterfile (and nothing more) to the current printout. Alas, the Acorn LJ printer drivers don't support PDriver_Select/InsertIllustration. I'm including everything from the first ESC,*,r,... (raster graphics on, inclusive) to the last 0x0c (page feed, exclusive). This system works fine with all the LJ printer drivers included in Printers 1.52.

abort_printing(...): In case anything went wrong during printing...

diagram(char *diagname, diagtrafo *dt): the actual printing code. It prints out the diagram with the filename diagname, using the transformations specified in dt, which include scaling in both directions as well as cropping from all sides.

One word concerning cropping in diagram: you could interpret cropping as simply removing parts of the diagram but keeping the position unchanged. I thought this was the way it was supposed to be at first but DVIview simply moves the entire image so I added support for this mode as well; define CROP_MODE_DVIVIEW and you'll get cropping like in DVIview otherwise the other variety is selected. The binary included in this release was compiled with CROP_MODE_DVIVIEW defined.

8.3 Parsing Diagrams

Parsing the \special command including the DVIview_diagram or drawfile commands is done in the procedure ParseDiagram which is called from DoSpecial, both in the file dvimisc.c. The reason for not using the existing DoSpecial is that the format of these \special commands differs quite a bit from the one assumed for the other \special commands supported. ParseDiagram will read all the parameters, call diagram with these and return _TRUE if the \special command was related to diagram printing, _FALSE otherwise.

Reading the arguments I allow somewhat more variations than DVIview does. Usually everything can be separated by ",", "=" or " ".

8.4 OSlib

Especially in `diagrams` I've made excessive use of OSlib. However GCC doesn't like some of the OSlib headers, so I had to rip out the bits I actually needed. I hope the authors of OSlib won't mind too much but that was the only way to do it. Besides I don't see much point in including half of the OSlib headers just because I want to use calls to the drawfile module.

8.5 The drawfile module

Another incompability, this time it's the linker `drlink` used by `gcc`. It can't read the AOF format used in Acorn's drawfile distribution, so I wrote short assembler padding functions myself, which are found in `drawfile.s`.

8.6 Type discrepancies

I've had to change some types a little. First of all, the original release of **dvilj** declared `bool` as `char`. OSlib, however, uses `unsigned int` which is longer so to avoid certain chaos I had to adapt `bool` to an `int`-sized datum.
Another change was made with OSlib's file descriptor type, `os_f`. This is originally `unsigned char` but was changed to `int`, the reason being **dvilj** requires a bigger filetype because it uses special conventions for file descriptors stored in a font descriptor struct:

- `NULL` means there hasn't been an attempt made to open the font yet.
- -1 means the font file doesn't exist.
- All other values are interpreted as legal file descriptors.

This is OK since with APCS all non-floating point arguments are passed as word-sized data. The only occasion where problems could occur is when you're passing an `os_f *` because in that case the top 3 bytes are undefined. That's why I'm initializing the file descriptor with 0 when I have to do this.

8.7 Makefiles

I have provided makefiles for GCC (`mfile`) as well as Acorn C (`makefile`). I'm not sure whether the Acorn C one works since I don't own this compiler, but I can see no apparent reason why it shouldn't.

8.8 The end

Programming is like sex: one mistake and you have to support for a lifetime.