The EPL5700L printer driver code

Hin-Tak Leung

23th February,2003

Contents

1 Introduction

Important: Much of this document is getting out-dated and unmaintained. Please refer to the source code for definitive information.

This document details everything I learn of how to drive the Epson printer EPL 5700L in the last three years since I bought one.

It has been confirmed that the 5700L 5800L, and 5900L shares the same core band/stripe compression algorithm, but have rather different job header, page header stripe header and page footers. Much of the information is shared with EPL 5800L, 5900L, and the difference will be noted. There is also a new model 6100L sold in Europe, which may be related.

There is a lot of hex codes and a lot of maths in this document. If you don't like it, turn away now…

This document was last updated on February 23, 2003.

No guarantee to the correctness to any part of this document. Use at your own risk.

1.1 How can one tell if one's printer may be compatible?

I have a EPL-5700L from UK, It has been confirmed that the 5800L and 5900L, sold in the rest of Europe and elsewhere (Germany, Italy, and Taiwan) are similar enough. Some newer models, such as 6100L, might be related to this family.

Marking at the back of my EPL-5700L: **MODEL L270D**.

The Reference guide for 5700L, titled "EPL-5700L/EPL-5700i", is shared with the 5700i (sold in US?), so that might be the same printer branded in a different name.

Note that some "EPL-5x00", "EPL-5x00N", "EPL-5x00-PS" understands ESC/P (Epson's printer language), PCL (HP's printer language), or Postscript (Adobe's printer language), and you are better off using ECS/P, PCL, or Postscript to drive these printers.

The compression algorithm shared by the 5700L, 5800L, 5900L generates these 26 bytes for an entirely white area (e.g. top of a blank page), which can be regarded as a signature for which this driver applies:

```
a0 1d 74 03 0e 80 01 d0
40 3a e8 07 1d 00 03 a0
80 74 d0 0e 3a 01 07 40
00 e8
```

It starts almost a hundred byte into the spool file, for the 5900L (which has the longest header).

Note that if you have Win2k instead of Win98se, you need to disable "enhanced printer support" or something like that in the printer driver control panel to see the actual spool file. Otherwise, Win2k seems to keep the pages as WMF (windows metafiles) until the last minute before conversion to something that the printer understands. If one pauses printing, the spool files are kept as "sp00001.spl", etc (the number is reset to start from 1 every time a windows box reboots) in the system spool directory under win98 or win2k. Just pause the printer (or disconnect the parallel cable), print and search for "*.spl".

1.2 How did I get these details?

Ghostsript on MS Windows can use the GDI sub system of the host and print to any printer that the OS itself knows about.

Win32 Ghostscript also has a print script which is batch-drivable to print-spool thousands of postscript files.

So what I can do, is to draw some simple lines and shapes with `xfig`, export as postscript, and print and collect the spool files and examine them. For example, I have a little perl program which generates postscript files with one single horizontal line of increasing length at steps of 600th of an inch. So I have a few thousands of spool files to analyse.

It is a lot of guess work. The major vector-based driving languages are Postscript (Adobe), PCL (HP), ESC/Page (Epson), LIPS (Canon). Tried all of them and none of them works. The specification for Postcript, PCL and ESC/Page are very well-documented and publicly available - and I have them somewhere.

I don't think Epson would choose to invent one more vector language (instead of re-using their own ESC/Page, or just use postscript or PCL), so the EPL-5700L has to take some sort of compressed bitmap, if it isn't fitted with much intelligence. Apparently it can be fitted with a PCL chip and/or a postscript chip to make it understand PCL and/or postscript, and the EPL-5700 (without L) does understand PCL according to the spec. So I tried printing simple pages, like a blank page with only a page number at the bottom, with one single horizontal line, etc and start from there. Over time, by looking at a lot of spool files of simple documents that I come up with, I gained a certain understanding of how it works.

2 The structure of a EPL-5700L print job

The structure of a typical print job is as shown in Figure 1. Please have a look at the figure quickly to get a mental picture.

Part	size (Byte)
Job Header	8
Page Header	25
Stripe Header	7
Page Footer	2
Job Footer	2

Table 1: Sizes of the components of a print job

The details of individual parts are described below.

2.1 Job Header

A total of 8 bytes (I include the off-set because I get confused sometimes myself as my tests are all written in C/Perl for which the first byte is byte 0). See table 2 for details.

Byte	off set	description	values
1	0x00	unknown	0x00
2	0x01	unknown	0x00
3	0x02	resolution related	0,1
4	0x03	resolution related	0,1
5	0x04	RITech status: off = 0, on = 1 (default)	0,1
6	0x05	Toner save: off = 0 (default), on = 1	0,1
7	0x06	Paper type: Normal = 0 (default) Thick (N) = 2, Thick (W) = 1, transparency = 3	0,1,2,3
8	0x07	Density: density 1 = 1, density 3 = 3 (default), density 5 = 5,	1,2,3,4,5

Table 2: Job Header

RITech[1].

Thick paper means 90-163 g/m^2; Thick Narrow (< 188 mm wide), Thick Wide (> 188 mm wide).

Most of it corresponds to choices in the Win32 print driver interface. There is no header field corresponding to “skip blank pages”, as expected.

The resolution-related byte 3 and 4 are used as in table 3 (oh yes, the maximum resolution of the printer is 1200dpi x 600dpi, which is called “1200dpi Class” in the printer control panel under MS Windows):

Byte 3	Byte 4	Description
0x00	0x00	300 x 300 (300dpi)
0x00	0x01	600 x 300 (600dpi Class)
0x01	0x00	600 x 600 (600dpi)
0x01	0x01	1200 x 600 (1200dpi Class)

Table 3: Job Header resolution usage

[1] Resolution Improvement Technology

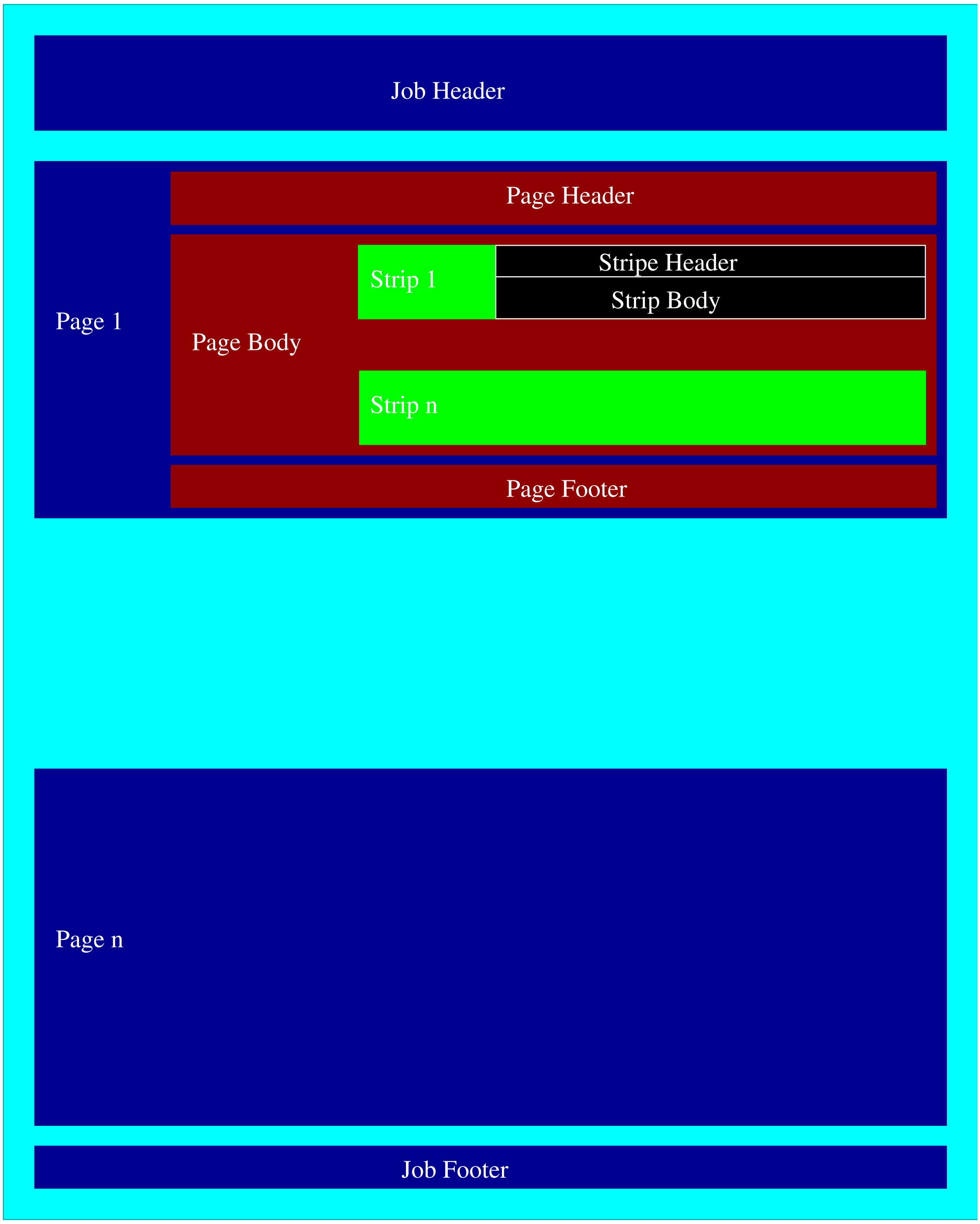

Figure 1: The structure of a typical print job

2.2 Page Header

The Page Header is 25 bytes, and the Page Footer is 2 bytes.

A page is divided into horizontal stripes of width 64 pixels each. So for example, an A4 portrait page at 600dpi horizontal and 300dpi vertical contains 54 stripes.

The math is like this: 300x 11.7inches = 3508, and 3508 /64 = 54.8 stripes, and remember an area of about 4mm at the top and the bottom is not printable.

Similar consideration applies for the horizontal and vertical pixel counts. They seems to be just the measurement of the paper size concerned minus the unprintable side margin.

The page header details is in table 4.

Byte	Off set	Off-set from job start	Description	value
1	0x00	0x0a		0x02
2	0x01	0x0b		0x00
3	0x02	0x0a	paper size code	
4	0x03	0x0b	unknown	0x40
5,6	0x04, 0x05	0x0c, 0x0d,	horizontal pixel count (/8, rounded up *4)	
7,8,9,10		0x0e,0f, 10, 11	unknown	0x00
11,12		0x12, 13	vertical pixel count	
13,14		0x14,15	horizontal pixel count	
15		0x16	unknown	0x00
16		0x17	Stripe count per page	
17		0x18	Tray selection: MP = 0, auto = 0xff (default)	0x00, 0xff
18		0x19	unknown	0x00
19		0x1a	Number of copies: 1 (default)	0x01, etc
20		0x1b	unknown	ff
21		0x1c	Avoid Page Error: 0xfe = off (default), 0xff = on	0xfe,0xff
22-25		0x1d to 0x20	Only used for Custom paper size	00 00 00 00

Table 4: Page header details

Byte 3 indicates the paper type and seems to be resolution independent (i.e. the code for A4 paper is fixed, regardless of printing at 600x300 or 1200x600). It seems to follow some kind of convention, so it might have been taken from an ISO specification table for paper sizes or just some Epson internal convention.

The others are all resolution dependent. See table 5 for how the paper size and pixel count bytes are used.

The horizontal pixel code at offset 0x0c, 0x0d is pixel count divided by 8 and then rounded up to multiple of 4 of offset 0x14, 0x15 . (i. e. rounded up to the next 32-bit boundary, converted to bytes) They seems to be simple pixel counts, most significant byte first.

TODO: I believe the printer only needs the paper size selection (byte 3), horizontal pixel count /8 *4, and the stripe count. I wonder what happens if the printer encounter inconsistent parameters. (i.e. when byte 5,6 don't agree with byte 13,14, or byte 11,12 don't agree with byte 16.

2.2.1 Custom Paper Size

Here is Custom Paper size (4.32 inch x 6.78 inch) compared with A4 (8.26 inch x 11.69 inch):

```
0e 40 02 54 00 00   00 00 0d 50 12 98 00 36   ff 00 01 ff fe 00 00 00   00
ff 40 01 2c 00 00   00 00 07 8d 09 56 00 1f   ff 00 01 ff fe 00 6d 00   ac
```

That last 4 bytes seems to be the custom paper size in mm (4.32 inch = 109mm, 6.78 inch = 172mm).

According to the windows GUI interface, custom paper size can only be within the parameters shown in Table 6. The minima are probably governed by physical distance between rollers, etc with the printer, while the max are by physical contraints of the paper tray and paper path.

2.3 Page Footer

See table 8 for the page footer.

0a	0c	0d	12	13	14	15	17	**Paper selection, off-set from job beginning**
0e	01	2c	0d	50	09	4c	36	A4
0f	00	d0	09	4c	06	70	26	A5
19	01	04	0b	78	08	02	2e	B5
1e	01	34	0c	80	09	92	32	LT
1f	00	c4	09	92	06	0e	27	HLT
20	01	34	10	04	09	92	41	LGL
21	01	04	0b	ea	08	1b	30	EXE
22	01	34	0e	d8	09	92	3c	GLG
23	01	20	0b	ea	08	fc	30	GLT
25	01	2c	0e	d6	09	4c	3c	F4
50	00	88	08	66	04	26	22	MON
51	00	90	0a	be	04	71	2b	C10
5a	00	98	09	c2	04	af	28	DL
5b	00	e4	0a	2c	07	15	29	C5
5c	00	9c	07	15	04	de	1d	C6
63	01	f0	0b	24	0f	74	2d	IB5
	01	2c	0d	50	09	4c	36	A4 300
	02	54	0d	50	12	98	36	A4 600c
	02	54	1a	a0	12	98	6b	A4 600
	04	a8	1a	a0	25	30	6b	A4 1200c
	01	34	0c	80	09	92	32	LT 300
	02	68	0c	80	13	24	32	LT 600c
	02	68	19	00	13	24	64	LT 600
	04	cc	19	00	26	48	64	LT 1200c

Table 5: Paper size and resolution (300x300 unless otherwise stated)

Dimension	min (cm)	max (cm)	min (inch)	max (inch)
Width	9.01	21.59	3.55	8.50
Height	14.80	35.56	5.83	13.99

Table 6: Custom paper size limits

Dimension	min (0.1mm)	max (0.1mm)
Width	920	2160
Height	1450	3560

Table 7: Custom paper size limits - according to the dll `eptrua1e.dll`

value	**Description**
03 00	Page ends

Table 8: The Structure of the Page Footer

2.4 Job Footer

See table 9 for the job footer.

value	Description
01 00	Job ends

Table 9: The Structure of the Job Footer

2.5 Stripe Header

Note - I always write my 1's and 0's in decoding order i.e. LSB first - this is unusual.

The 7-byte Stripe Header is simply `04 00 01 00` followed by the stripe byte count, most singnificant byte first.

An empty stripe is 64 groups of `101110 0000000` [2], which is 104 bytes long; so a blank A4 page (the smallest A4 print job) at 600dpi x 300 dpi is 10 (job header) + 23 (page header) + 54 x (7 + 104) +4 = 6031 byte long. The stripe header for a blank stripe is `04 00 01 00 00 00 68` (0x68 = 104).

The worst case scenario, 1200dpi horizontal with random noises, contains 1200 x 9 inches x 64 pixels ≈ 700,000 per stripe, or would take about 90,000 bytes to encode literally; so 3 bytes for byte count should be enough.

2.6 The 5700L command set

If one collects the first two bytes of every structure, it becomes obvious that there is a pattern. (table 10).

value	Description
00 00	Job starts
01 00	Job ends
02 00	Page starts
03 00	Page ends
04 00	Strip starts
05 00	(USB only) unknown — 2nd before job
06 00	(USB only) unknown — 1st before job
07 00	(USB only) unknown — between pages
08 00	(USB only) unknown — while idle

Table 10: The 5700L command set

3 The Stripe compression algorithm

Much of the information in this section is due to Roberto Ragusa. Many thanks.

The strip content is a 16-bit bit-stream with the most significant byte first, so to understand it properly, one has to read the stream like this: {byte 2 bit 1, bit 2 ,. . . , bit 8 }, {byte 1 bit 1, bit 2 ,. . . , bit 8 }, {byte 4 bit 1, bit 2 ,. . . , bit 8 }, {byte 3 bit 1, bit 2 ,. . . , bit 8 }, etc. It is padded at the end to 16-bit boundary so the stripe byte count is always even.

The compression algorithm works like this: Compare with the previous row, and see how many bytes (8-bit) are the same. If the current byte is different from the byte above, see if it is the same as the byte before, 2 byte before, and 3 byte before. We count the number of consecutive bytes of sameness for these cases. If the current byte is neither the same as the one above or the 3 bytes before, then it is encoded as a literal. However, the literal bytes are cached in a 16-element cache table, with a First-In-First-Out policy, and the algorithm tries to see if a literal has already been seen before.

One of the clever things about the algorithm is that the compare-with-byte-before scheme sees if there is periodicity of 8-bit, 16-bit or 24-bit in the input. The last one is most interesting, as it means 3-bit, 6-bit period

[2] The line termination code - more about this in the compression algorithm section.

are also catered for, as 3-bit and 6-bit period repeats every 24-bit also. The algorithm exploits sameness between rows, and also periodicity within a row of 8-bit, 16-bit or 24-bit.

At the beginning of each stripe, the cache is pre-initialized (somewhat arbitrarily) to `0x00` to `0x0f`. This is sub-optimal, as pre-initializing to runs of black and white bits of various length should be more likely to keep cache modification a minimum (and increases the compression efficiency).

3.1 The op codes

See table 11 for the bit-patterns emitted by the encoder/compressor corresponding to the different operations. The decode tree is in figure 2.

Description	**bit pattern**	**Followed by**
Look up from Cache	00	4-bit index to cache
New Literal	01	8-bit literal
Copy from above	10	count code (see next table)
Copy 1 byte	110	count code (see next table)
Copy 2 bytes before	1110	count code (see next table)
COpy 3 bytes before	1111	count code (see next table)

Table 11: The Op codes

3.2 The Run Length Encoding

The run-length count (for how many byte to copy from the previous row or previous bytes on the same row) is encoded as in table 12.

count	**bit pattern**
1	0
2	10
3	1100
4	1101
5	11110
6	111110
7	111111
$8 \leq x < 128$	1110 $\langle$*7-bit*$\rangle$
$128 \leq x < 256$	1110 $\langle$*7-bit*$\rangle$ $\langle$*7-bit*$\rangle$
$256 \leq x < 384$	1110 $\langle$*7-bit*$\rangle$ $\langle$*7-bit*$\rangle$ $\langle$*7-bit*$\rangle$
$384 \leq x < 512$	1110 $\langle$*7-bit*$\rangle$ $\langle$*7-bit*$\rangle$ $\langle$*7-bit*$\rangle$ $\langle$*7-bit*$\rangle$
$512 \leq x < 640$	1110 $\langle$*7-bit*$\rangle$ $\langle$*7-bit*$\rangle$ $\langle$*7-bit*$\rangle$ $\langle$*7-bit*$\rangle$ $\langle$*7-bit*$\rangle$

Table 12: The Run-Length Code

Since the first 7-bit code can never be below 7, a 7-bit code of zero is used for line termination. i.e. the bit code `10 1110 0000000` equals “copy the entire row above, however many byte it is”. Because it requires only one group of 7-bit rather than a few groups of 7-bit for an explicit byte count to the end, this saves a few bits , when the rest of a row is “uninteresting”.

3.3 An example

An example of a shape I drew with `xfig` and printed out is shown in figure 4. The corresponding bit code for the first non-trivial strip (3rd strip) is shown in table 13, for A4 at 300 x 300 dpi. When xfig exports to pdf it centres the shape so it has moved somewhat sideways towards the centre of the paper.

It is the 3rd stripe, 2x 64 + 48 = 176 pixels, or just over half an inch from the top of the printable area of a page. The first non-trivial line is Copy 108 x 8 pixels (about 3 inches), 1 literal black byte, copy that black byte 55 times, ie. 59x8 pixels (an inch and half), and half a literal black byte after. Thereafter, it is just copying 165/164/163 bytes and literal 1 byte to get a shrinking black line scan.

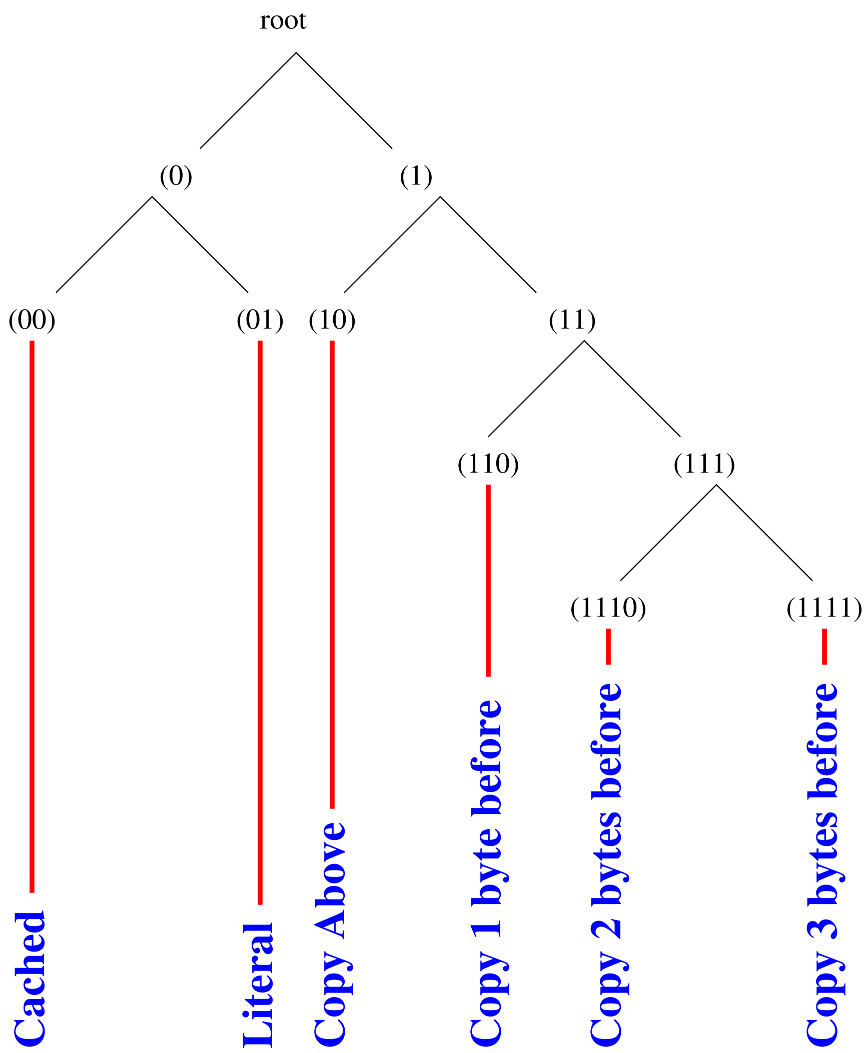

Figure 2: The decision tree for the Op Code

bit code	**Description**
`10 1110 0000000` x 48	copy (blank) row abow x 48
`10 1110 0011011 00 1110`	copy 108, cache[7](⟨= `11100000`)
`01 11111111`	literal 1 (=⟩ cache[0])
`110 1110 1110110 01 00000001 10 1110 0000000`	copy-left 55, literal 1 (=⟩ cache[1]), line end
`10 1110 1111111 0110010 01 00000000 10 1110 0000000`	copy 165, literal 1 (=⟩ cache[2]), line end
`10 1110 1111111 1010010 01 01111111 10 1110 0000000`	copy 164, literal 1 (=⟩ cache[3]), line end
`10 1110 1111111 1010010 01 00111111 10 1110 0000000`	copy 164, literal 1 (=⟩ cache[4]), line end
`10 1110 1111111 1010010 01 00011111 10 1110 0000000`	copy 164, literal 1 (=⟩ cache[5]), line end
`10 1110 1111111 1010010 01 00001111 10 1110 0000000`	copy 164, literal 1 (=⟩ cache[6]), line end
`10 1110 1111111 1010010 01 00000111 10 1110 0000000`	copy 164, literal 1 (=⟩ cache[7]), line end
`10 1110 1111111 1010010 01 00000011 10 1110 0000000`	copy 164, literal 1 (=⟩ cache[8]), line end
`10 1110 1111111 1010010 00 1000 10 1110 0000000`	copy 164, cache[1] ((⟨= `00000001`), line end
`10 1110 1111111 1010010 00 0100 10 1110 0000000`	copy 164, cache[2] ((⟨= `00000000`), line end
`10 1110 1111111 0010010 00 1100 10 1110 0000000`	copy 163, cache[3] ((⟨= `01111111`), line end
`10 1110 1111111 0010010 00 0010 10 1110 0000000`	copy 163, cache[4] ((⟨= `00111111`), line end
`10 1110 1111111 0010010 00 1010 10 1110 0000000`	copy 163, cache[5] ((⟨= `00011111`), line end
`10 1110 1111111 0010010 00 0110 10 1110 0000000`	copy 163, cache[6] ((⟨= `00001111`), line end
`10 1110 1111111 0010010 00 1110 10 1110 0000000`	copy 163, cache[7] ((⟨= `00000111`), line end
`10 1110 1111111 0010010 00 0001 10 1110 0000000`	copy 163, cache[8] ((⟨= `00000011`), line end
`000000000`	padding

Table 13: Bit code for the top part of the triangular shape

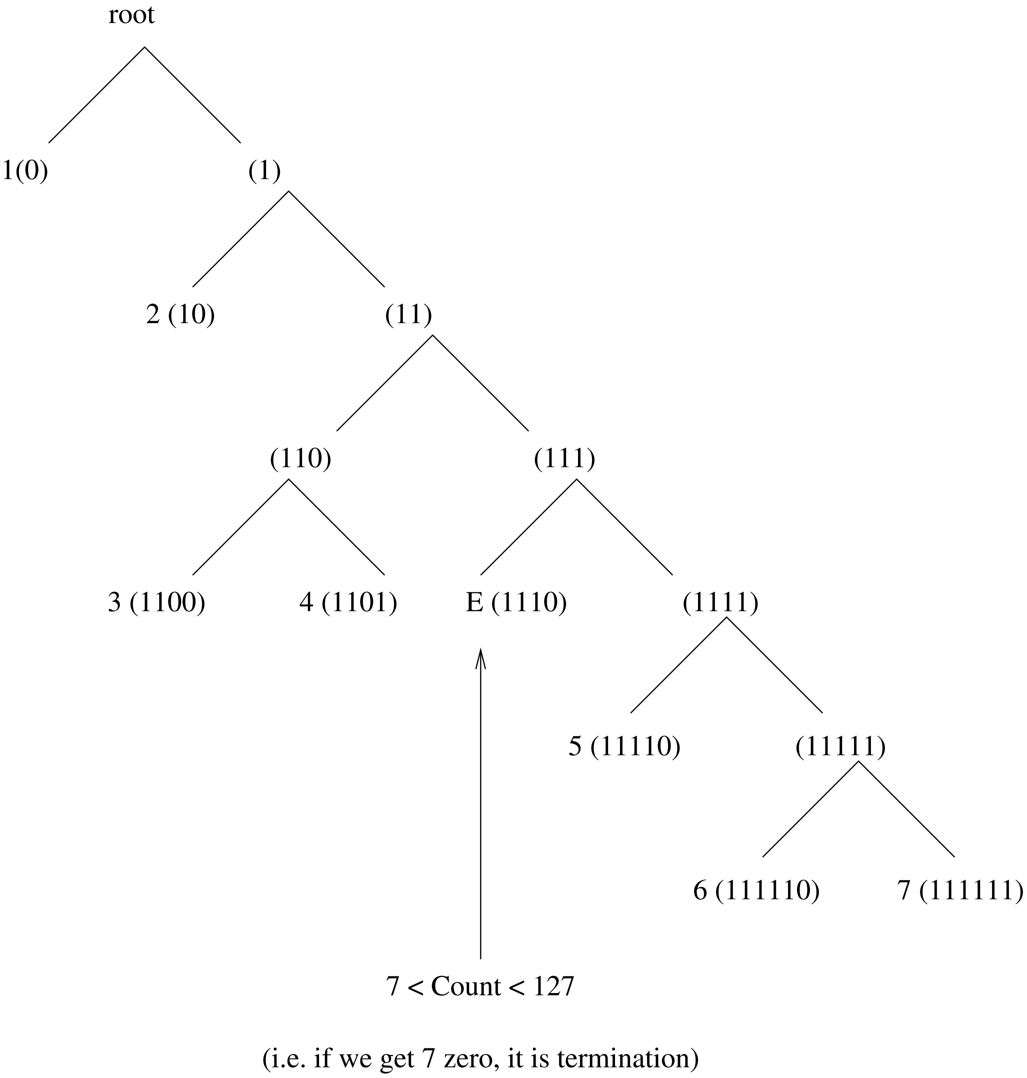

Figure 3: The decision tree for the Run Length Code

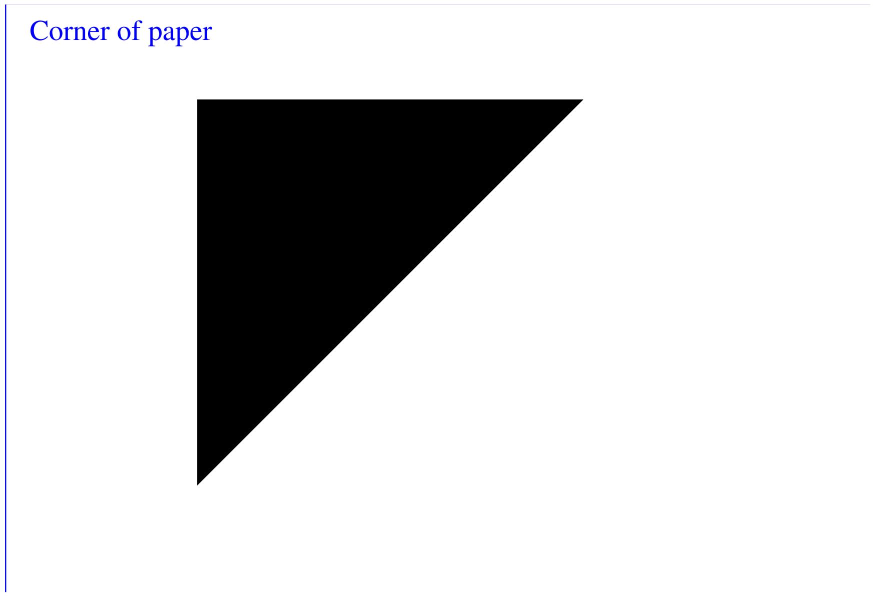

Figure 4: An example of a test figure

3.4 Other snipplets of information

I have a couple of small programs for bit-disassembling strips and strip-assembling from bits with the double-byte swapping and padding to 16-bit boundary included. So I can disassemble, modify, assemble, `cat ⟩ /dev/lp0` to see what it looks like.

4 Difference between EPL-5700L, 5800L, 5900L

Broadly speaking, the 5800L and 5900L have `0x1d`, then a byte count in ascii, then `eps{I` prepended to every structure of a print job, compared to the 5700L. They also have some extra EJL commands of the form `@EJL ` , then the command, then `<LF>` before and after the job. The 5900L has more EJL commands compared to the 5800L.

```
0x1d <count> eps{I
```

See the source code for most up-to-date details.

5 Difference between the USB interface and the parallel port interface

The EPL-5900L is reported to work uni-directionally via USB just like the parallel port code. The case for 5800L is not yet reported.

The EPL-5700L definitely does not work uni-directionally via USB. Every structure of the job requires a read from the printer, except the stripes. Also, it doesn't buffer input very well; the input is required to be well-timed. The data read back is at least 15 bytes. The 15th byte is a byte count for how many to follow. The number of bytes to follow after 15th is related to the stage of the print job.

Here is an example, of the reply from job header:

```
00 00 00 00 1f ef 38 00   00 00 00 00 00 02 00 ...
```

Byte 1,2 always reflect the original command.

Byte 5th — 7th of the 15 bytes seems to be some kind of temperature/charge meter, which stays contant while the printer is idle, drops right after a job, and come back up to the previous level gradually after a print job.

Byte 8 is some kind of readiness indicator. It changes from `0x00`, to `0x10` after the job header is submitted, and stays at `0x10` until the Byte 5th — 7th rises back to the pre-job level, at which point `0x01` is replied to the polling, to which the driver is supposed to reponds with different query, at which point it goes back to `0x00`.

EPL-5800L has a minimum 18-byte reply. The 18-th byte is also a count. There are also correponding structure for the temperature/charge meter, and the readiness indicator.

6 Intellectual Property Right Issues

I have to say in advance that, I have every intention of honouring the IPR of Epson; I know it takes years of work of many talented individuals to do the work they do. I would be happy if they provide a close-source driver, but I am stuck with a printer that I bought which I can't use under linux. I want to have my consumer rights of being able to use it to print documents from the OS of my choice. I was trained as a theorectical research physicist and my primary document preparation system is LaTeX based under linux/unix, and I don't touch MS Office if I can avoid it. In fact I have not *ever* used Word 7 and Word 8 for any document for which I am the starting author! I bought the printer mistakenly because the documentation of ghostscript has an error about 5700L being supported (the 5700, without L, is apparently drivable with PCL). Although to use the printer, I can generate a postscript file, reboot my dual-boot box to windows and print it via win32 ghostscript's GDI driver, it is just too painful for every document I want to print. Hence this effort.

Epson Seiko has a patent application family which applies to the US, Europe and most of the civilized world which details a more primitive version of the compression algorithm. Therefore, the compression algorithm should not be used for any other purpose than for interacting with the Epson EPL printers.

6.1 The MS Windows driver

Under win98, there is a dll `eptcmpa0.dll` which has the following symbols (and corresponding clearly identifiable routine sections):

EPCompressBitsImage
EPCompressGlyph
EPCompressImage
EPExpandBitsRLE
EPExpandRLE
EPGetCompressBitsBufferSize
EPGetCompressBufferSize

Under Win2k, the `eptminb7.dll` contains these strings (but no identifiable starts and ends of routines):

EPCompressImage
EPGetCompressBufferSize

I had a quick look at the assembler dump of these (there is a few Win32 PE disasemblers which run under linux); they seem to have what I want, but bit-suffling in assembler is quite complicated and essentially pure bit-manipulation maths and one needs a lot of patience to read them…

I don't have any deep knowledge about MS windows programming (I am all unix based, almost exclusively). But maybe this information is useful for somebody who understands windows dll's know about calling conventions and the win32 printing sub system, etc (e.g. the wine people…). Both of the windows drivers come with about 30 dll's, so at least this narrows it down to one to save some investigative work.

It might be quite interesting to try to interface directly with the DLL via some Wine code snipplets to print. This is for x86 unices only, I guess.

Table 14 lists the DLL's which seem to contain the compression code. Only `EPTCMPA0.DLL` on a 5700L seem to have the expansion code.

7 Misc

7.1 On-line Resources

Most of these pages are in Japanese (I am Chinese, so I can read a good deal of written Japanese...), and they may be out-dated. I thought it would be useful to patch ghostscript with some extra drivers specific to the Japan locale

Location	MD5 sum	Size (byte)	Date Stamp
EPL-5700L/Driver/DISK2/WIN9X/EPTCMPA0.DLL	e6f0dee5281a2cbfba405ece14e8a02d	51200	Jul 2 1999
EPL-5700L/Driver/DISK4/WINNT40/EPTCMPA0.DLL	e6f0dee5281a2cbfba405ece14e8a02d	51200	Jul 2 1999
EPL-5700L/Driver/DISK4/WINNT40/EPTMINA3.DLL	f7b161623bb2434efab10b5dcd89f35b	40672	Jul 2 1999
EPL-5700L/Driver/DISK5/WIN2000/EPTMINB7.DLL	3fd07c7ae23fd67b3f09fff55b154401	32280	Dec 14 1999
EPL-5700L/EPL5700LNT4/WINNT40/EPTCMPA0.DLL	e6f0dee5281a2cbfba405ece14e8a02d	51200	Jul 3 1999
EPL-5700L/EPL5700LNT4/WINNT40/EPTMINA3.DLL	f7b161623bb2434efab10b5dcd89f35b	40672	Jul 3 1999
EPL-5700L/EPL5700LW2K/WIN2000/EPTMINB7.DLL	3fd07c7ae23fd67b3f09fff55b154401	32280	Dec 15 1999
EPL-5700L/EPL5700LW95/WIN9X/EPTCMPA0.DLL	e6f0dee5281a2cbfba405ece14e8a02d	51200	Jul 3 1999
EPL-5700L/EPL5700LW98_ME/WIN9X/EPTCMPA0.DLL	e6f0dee5281a2cbfba405ece14e8a02d	51200	Jul 3 1999
EPL-5800L/5800l_nt4_2_14dm/WINNT40/EPTMINB3.DLL	e615207b14bef94e2d892a56cad4ee22	43680	Aug 29 2000
EPL-5800L/5800l_w2k_2_14fm/WIN2000/EPTMINB7.DLL	7bbcc27af36f9f1c3d89aafbd49f6f83	33287	Aug 29 2000
EPL-5900L/WIN2000/EPTMINC7.DLL	2f6ef4c0a8ac2c4bd3106f761c5d4c38	118069	Jul 5 2001
EPL-5900L/WINNT40/EPTMINC3.DLL	c9ec07d1ea7a6492a136c1e38cd30f20	135072	Jul 5 2001

Table 14: DLL's which seems to contain the compression algorithm.

(Epson being Japanese and what not), but none of it worked (in mid-2000, after I got the printer and before I took a job where mostly MS windows is used).

The first one is in English, and the official Epson printer support for linux and is probably most useful, and some Epson employees see to be hanging around the forum, so posting to the forum there might get some attention... although they have explicitly say that the 5700L, 5800L, 5900L is windows and Mac only. I suppose if I provide this much detail here, it might pressure them into giving me some actual help.

- EPSON KOWA CORPORATION - linux driver forum
  http://www.epkowa.co.jp/english/linux_e/linux.html
- How to add printer device to gs
  http://www.ee.t.u-tokyo.ac.jp/~mita/FreeBSD/gsprinter.html
- Ghostscript 6.01 and GSview 2.9 J
  http://auemath.aichi-edu.ac.jp/~khotta/ghost/index.html
- gdevepag ver.3 http://www.humblesoft.com/gdevepag.html
- Software Archive
  http://www.tcp-ip.or.jp/~tagawa/archive/index.html
- Norihito Ohmori's WWW page
  http://www.bukka.p.chiba-u.ac.jp/~ohmori/
- gdevmd2k
  http://plaza26.mbn.or.jp/~higamasa/gdevmd2k/
- Ghostscript drivers
  http://unicorn.p.chiba-u.ac.jp/~ohmori/gs/
- Ghostscript drivers
  http://www.bukka.p.chiba-u.ac.jp/~ohmori/gs/
- Ghostscript driver for LIPS & ESC/Page & NPDL
  http://www.bukka.p.chiba-u.ac.jp/~ohmori/gs/Gdevlips.htm
- Fujitsu FMLBP 2xx driver for Ghostscript
  http://www1.freeweb.ne.jp/~nakayama/gdevfmlbp-120.html
- gswin5.50j information
  http://itohws03.ee.noda.sut.ac.jp/~matsuda/gswinj/gswin5j.html
- FORMPRINT for Linux(ESC/Page)
  http://www.vector.co.jp/soft/unix/hardware/se116245.html

7.2 The USB interface

This document details what is going through the parallel port. In fact, a spool file generated by the windows driver can be send by linux and print successfully like this (by the root user):

```
cat sp00001.prn > /dev/lp0
```

However, sending the code through `/dev/usb/lp0` doesn't work. There are various documentation on the net (just search for "Epson printer" and "USB" and "linux") which says that for Epson printers which has a USB interface, one has to put an ESP/Page Job Language header before the job like this (in hex) to enable the USB interface: Tried it already, and it won't work.

```
00 00 00
1b 01 40 45 4a 4c 20 31 32 38 34 2e 34 0a
40 45 4a 4c 20 20 20 20 20 0a
```

(This is actually the hex code for "EJL 1284.4 @EJL" with some extra null bytes, line feeds, carriage returns, spaces, etc).

It is quite simple to find out how the USB interface of the EPL5700L works - just use a USB snoop utility (c.f. the linux usb support page) to have a look at what goes through while a print job is going through a win32 host to the printer connected via USB. Apparently the 5700L expects true bi-directional communication. It feeds back at least 15-bytes to be read back by the driver for every part of the print job, except the stripes.

EPL5900L known to work uni-directionally via the`/dev/usb/lp0` device. No change needed for EPL5900L.

7.2.1 5700L USB info

```
root@pc7221:/proc/bus/usb# cat devices
<irrelevant usb controller info snipped>
T:  Bus=01 Lev=01 Prnt=01 Port=00 Cnt=01 Dev#=  2 Spd=12  MxCh= 0
D:  Ver= 1.00 Cls=07(print) Sub=01 Prot=02 MxPS=64 #Cfgs=  1
P:  Vendor=04b8 ProdID=0001 Rev= 1.00
S:  Manufacturer=EPSON
S:  Product=USB Printer
C:* #Ifs= 1 Cfg#= 1 Atr=40 MxPwr=  2mA
I:  If#= 0 Alt= 0 #EPs= 2 Cls=07(print) Sub=01 Prot=02 Driver=usblp
E:  Ad=01(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=82(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
root@pc7221:/proc/bus/usb#
```

7.2.2 5800L USB info

```
T:  Bus=02 Lev=01 Prnt=01 Port=01 Cnt=01 Dev#=  2 Spd=12  MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=04b8 ProdID=0005 Rev= 1.00
S:  Manufacturer=EPSON
S:  Product=USB Printer
S:  SerialNumber=XXXXXXXXXXXXXXXXXX
C:* #Ifs= 1 Cfg#= 1 Atr=c0 MxPwr=  2mA
I:  If#= 0 Alt= 0 #EPs= 2 Cls=07(print) Sub=01 Prot=02 Driver=usblp
E:  Ad=01(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=82(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
```

7.2.3 5900L USB info

```
T:  Bus=01 Lev=01 Prnt=01 Port=01 Cnt=01 Dev#=  2 Spd=12  MxCh= 0
D:  Ver= 1.10 Cls=00(>ifc ) Sub=00 Prot=00 MxPS=64 #Cfgs=  1
P:  Vendor=04b8 ProdID=0005 Rev= 1.00
S:  Manufacturer=EPSON
S:  Product=USB Printer
S:  SerialNumber=XXXXXXXXXXXXXXXXXX
```

```
C:* #Ifs= 1 Cfg#= 1 Atr=c0 MxPwr=  2mA
I:  If#= 0 Alt= 0 #EPs= 2 Cls=07(print) Sub=01 Prot=02 Driver=usblp
E:  Ad=01(O) Atr=02(Bulk) MxPS=  64 Ivl=0ms
E:  Ad=82(I) Atr=02(Bulk) MxPS=  64 Ivl=0ms
```

7.2.4 5700L USB misc.

A normal 15-byte reply from a 5700L consists of the 2-byte command, then

```
00 00 1F EF A0 00 00 00 00 00 00 00
```

Then a byte count.

Number	Description	Value
1,2	Command reflection	
3,4	unknown	0, 0
5,6,7	temperature/charge	
8	readiness (0x00 = idle, 0x10 = ready, 0x01 = standby)	
9,10	unknown	0, 0
11	paper jam, no paper (0x00 = okay, 0x10 = paper propblem)	
12,13	unknown	0, 0
14	accumulated error count (reset to zero at power off)	
15	byte count to follow	

The **charge** item drop from `1F EF A0` to `17 E6 E2` and gradually comes up again; idle at `1F EF 38`.

Typical replies (after byte 15):

```
06 00 - 02 03 FE 00
05 00 - 00 1F FF F8 00 30
07 00 - 01 31 08 B8 02 00 00 08 4A 41 59 5F FF
```

The last few bytes of reply from 07 00 changes somewhat (increasing, decreasing, going up and down respectively):

```
08 38 42 59 62 FF
08 39 42 59 61 FF
08 3A 42 59 60 FF
08 3B 42 59 60 FF
08 3C 42 59 5F FF
08 3D 42 59 5E FF
08 3E 42 59 5E FF
08 3F 42 59 5D FF
08 40 42 59 5C FF
08 41 41 59 5B FF
08 42 41 59 5B FF
08 43 41 59 5A FF
08 43 41 59 64 FF
08 44 41 59 63 FF
08 45 41 59 62 FF
08 47 41 59 61 FF
08 48 41 59 60 FF
08 49 41 59 60 FF
08 4A 41 59 5F FF
```

Some of it could be: Time to go standby, Toner Level, Photoconductor Life, total pages printed, installed memory, firmware version, etc. Paper size and amount in the feeder and the MP tray.

7.3 Specifications

Extracted from various Epson Documentations.

7.3.1 Ready and Error lights

The ready (green) and error (red) lights on the top of the printer indicate the printer status. Whether the lights are flashing or not indicates different things, as described below. More detailed information can be found in the Alert window in the EPSON Status Monitor 3.

Ready Light (green)	Error Light (red)	Printer Status
off	off	Printer is off.
on	off	Printer is ready.
Flashing	off	Printer is warming up or receiving data.
off	Flashing	Either the printer is out of paper or there is an error that can be easily remedied.
off	on	This is an error that requires a service call. Alternatively, try turning off the printer and then turning it back on. If this does not remedy the error, contact your dealer or a qualified service person.

The 5800L documentation have these extra entries:

Ready Light (green)	Error Light (red)	Printer Status
Flashing	on	Either the printer is out of paper or there is an error that can be easily remedied.
Flashing alternatively		This is an error that requires a service call. Alternatively, try turning off the printer and then turning it back on. If this does not remedy the error, contact your dealer or a qualified service person.

7.3.2 Using the DMA

Faster printing with a Direct Memory Access (DMA) is possible if your printer is connected to your computer through an Extended Capability Port (ECP), a type of parallel port with expanded specifications. With the DMA and the ECP, you can send print-job data directly to the printer without going through the CPU, which increases the effectiveness of data flow. Before using DMA to transmit a print job confirm the following points.

- Check if your computer supports DMA. To confirm if your computer has an ECP chip installed, contact the manufacturer or look in the computerbegs documentation.
- Confirm the parallel port setting is ECP or ENHANCED through the BIOS Setup. Refer to your computerbegs documentation to set the BIOS. However, before setting the BIOS, uninstall the EPL-5700L/EPL-5700i printer driver. After setting the BIOS, re-install the printer driver.

7.3.3 Confirming if you can use DMA

Using DMA increases your printing speed. You can confirm if your computer can use DMA by checking the Optional Settings tab under Properties in your Printer driver.

7.3.4 Memory Module

By installing a Single In-line Memory Module (SIMM), you can increase printer memory up to 13 MB (printer default 2MB + optional 4, 8, 16 or 32 MB). You may want to add additional memory if you are having difficulty printing complex graphics or if you regularly use numerous downloaded fonts. [3] [4]

7.3.5 Status Monitor

Printer image: The image at the upper left shows the printer status graphically.

Text box: The text box next to the printer image displays the current status of the printer. When a problem occurs, the most likely solution is displayed. (Normally “Ready to print”; e.g. “Top cover is open”).

OK button: Click OK to close the dialog box.

[3] The maximum amount of memory is 13 MB. Even though a 16 or 32 MB SIMM is installed, only 13 MB is accessible.

[4] After installing the printer driver, you can confirm the optional memory is installed correctly. Windows users can check in the Printer Driver by selecting the Optional Settings tab under Properties, and Macintosh users can check the memory installation in the EPSON Status Monitor.

Close button: Click Close to close the dialog box. Simple display Reduces the display and shows only the status details screen.

Paper: Displays the paper size and approximate amount of remaining paper in each paper source. Optional cassettes are displayed only if they are installed. (“MP Tray”, “Lower Cassette”).

Toner: Displays the amount of toner remaining. The toner icon flashes when the toner is low (10 percent or less).

Photoconductor Unit Life: Displays the Photoconductor unitbegs remaining functional life.

The status alert window can pop up if there is anything wrong with the printer.

7.3.6 Status Monitor Preferences

Notification
Paper Size check before printing
Paper Size check after printing
Toner Low
Other warnings (e.g. Top cover open)

7.3.7 Paper size

Paper	A4 (210 mm x 297 mm) A5 (148 mm x 210 mm) B5 (182 mm x 257 mm) Letter [LT] 216 x 279 mm (8 1/2 in. x 11 in.) Half Letter [HLT] 140 x 216 mm (5 1/2 x 8 1/2 in.) Legal [LGL] 216 x 356 mm (8 1/2 x 14 in.) Executive [EXE] 184 x 267 mm (7 1/4 x 10 1/2 in.) Government Legal [GLG] 216 x 330 mm (8 1/2 x 13 in.) Government Letter [GLT] 203 x 267 mm (8 x 10 1/2 in.) F4 (210 mm x 330 mm), 8.2 x 13 in Custom/User Defined (76.2 mm x 127 mm to 215.9 mm x 355.6 mm, 3 x 5 in. to 8.5 x 14 in.)
Envelopes:	Monarch [MON] 98 x 191 mm (3 7/8 x 7 1/2 in.) Commercial 10 [C10] 105 x 241 mm (4 1/8 x 9 1/2 in.) DL (110 mm x 220 mm), 4.3 x 8.7 in C5 (162 mm x 229 mm), 6.4 x 9 in C6 (114 mm x 162 mm), 4.5 x 6.4 in International B5 (176 mm x 250 mm), 7 x 9.9 in

Printable area: The area on a page contained within a 4 mm minimum margin on all sides

Printing method:	Laser beam scanning and dry electrophotographic process
Resolution:	600 x 600 dpi, 300 x 300 dpi
Printing speed:	Up to 8 pages per minute depending on the font and quantity of data
First print:	Less than 19 seconds with A4/letter
Warm-up time:	About 20 seconds at normal temperature
Paper feed:	Automatic
Paper feed alignment:	Center alignment for all sizes
Input paper supply (75g/m paper):	Up to 150 sheets in the tray Up to 500 sheets per optional 500-Sheet Lower Cassette Unit Up to 10 envelopes, depending on thickness
Paper output:	Face-down or face-up (optional) selection
Paper output capacity (75g/m paper):	Face-down 100 sheets Face-up 20 sheets with the optional Faceup Tray
RAM:	2 MB, expandable up to 13 MB
Durability:	5 years or 180,000 sheets, whichever comes first

The 5800L lists 18 seconds for the first print, 22 seconds for warm up time, and 10 pages per second.

Parallel port signal: High signal indicates that a feed jam occurred in the paper supply section, or that there is no paper in the paper-tray or cassette.

7.3.8 consumables

The number of pages you can print with a developer cartridge or a photoconductor unit varies depending on the type of printing. If you print a few pages at a time or print dense text exceeding the 5% print ratio, the unit may print fewer pages. The 5% print ratio is equivalent to printing double-spaced standard text.

Developer cartridge (S050010)

Storage temperature: 0 to 35 (32 to 95)

Storage humidity: 30 to 85% RH

Shelf life: 18 months after production Life: Up to 6000 pages under the following conditions: Letter or A4 size paper, continuous printing, and 5% print ratio

Photoconductor unit (S051055)

Storage temperature: 0 to 35 (32 to 95)

Storage humidity: 30 to 85% RH

Shelf life: 18 months after production Life: Up to 20000 pages under the following conditions: Letter or A4 size paper, continuous printing, and 5% print ratio

The 5800L uses the same part numbers for consumerable.

7.3.9 misc.

The dll `eptrua1e.dll` in the 5700L driver contains most of the user dialog strings for GUI.

The 5700L and 5800L windows drivers have an options for clearing the toner level and the OPC level by software, after a new toner cartridge or an new Photoconductor unit is installed; also for resetting the EEPROM on the printer.

The Standby mode saves energy by reducing power to the printer when the printer receives no data for 30 minutes. There is an option for enabling that, and also to changing the time to standby value to 5 minutes, 15 minutes, or 60 minutes.

The driver can see installed memory on the printer.