Printer Hacking

Gene Michael Stover

created Saturday, 2004 November 13
updated Tuesday, 2006 June 20

Contents

Chapter 1

What is this?

These are my own notes for programmers (mostly me). These are techniques for making printers do things & for programs which do things to printers.

"Did you said hacking?" Indeed. I'm usng the definition of hack from *The Jargon File* as maintained by Eric S. Raymond. [3] I guess I'm specifically using part 6 of that definition. Anyway, if you are looking for a way to crack the passwords on printers, go away. And if you are thinking "You can break into a printer", you're pretty much correct.

Chapter 2

Basics of modern printer technologies

Many (most?) modern LaserJet & similar printers have built-in web servers. You can point your web browser at them if you know their IP address, & you'll be looking at a web site which shows whether the printer is currently printing, how much paper & toner are loaded into it, & maybe other details such as the page counts, page description languages it understands, & network protocls it speaks. You can often change settings on the printer unless someone has set an administrator password. (By default, they almost never have an administrator password.)

Chapter 3

When you tell a printer to print

When you tell an application to print a document (or picture or file or whatever), here's what happens:

1. The application converts the document into a system-specific imaging language. Statements in this system-specific language might be expressed explicitly as data, or they could be implicit in interactions between the conversion library & the printer driver. However it's done in reality, the concept of the system-specific imaging language applies.

   On Microsloth Winders, the application obtains a Device Context (`HDC`) to the printer, then draws the image with the same API it used to draw on the screen.

   On unix-like systems, the system-specific imaging language is less standardized. Sometimes there isn't one, & the application is responsible for the conversion. In this case, the application will probably convert directly to a page description language.

2. The printer driver consumes statements in the system-specific imaging language & converts them to statements in a language the printer understands.

   If the application had to convert directly to a page description language, this might be a null-step. Then again, it might not. For example, an applicaition could have converted its document to PostScript, but the printer driver might convert that to PCL or some other language. Gnu Ghostscript is sometimes used in this way.

3. The printer driver sends the printer-specific page description language to the printer, possibly wrapped in some kind of job control language such as PJL.

It's probable that the printer driver doesn't communicate directly with the printer. In that caes, the printer driver probably gives the page description program (wrapped in job control statements) to a communication layer which is often called a *port driver*.

4. At last, the printer prints the document.

3.1 Page description languages

We use *page description languages* to tell printers what to draw. With a printer language, you (or, more likely, a program) can tell a printer to put *this much* toner of *this* colour at *that* location on the paper.

The two main page description languages are PostScript and PCL. In fact, they are the only page description languages of which I'm aware. I'm sure there are many other page description languages, most of them specific to particular printer models. Happily, they are on the wane, leaving us with two standard page description languages.

PostScript is a supremely cool language. It was created by Adobe in the 1980s, & Adobe still controls the language standard.

I love PostScript. [4]

PostScript is sort of like Forth. Implementations can be very efficient. It's portable. It's Turing-complete; with the proper I/O primitives, it could compute anything that higher level programming languages could compute. In fact, the PostScript language standard specifies a fairly versatile & complete set of I/O primitives for files, & many PostScript implementations include those I/O primitives.

If you want to learn PostScript, start with [1]. If you write a few programs with the information in that book & then find yourself wishing you had more complete information about the language, it's time to read [2].

PCL is another page description language. It originated at Hewlet Packard. I know very little about PCL, but from what I've seen, it's not nearly as neat-o as PostScript. From what I've seen, PCL is to PostScript as COBOL is to Lisp. *fixme: Where is the PCL language specification?*

Interestingly, the installable file for PostScript printer drivers for Microsloth Winders are less than half a megabyte, whereas the installable file for PCL printer drivers for Microsloth Winders area always considerably larger, often being something like five megabytes.

3.2 What about line printers? Dot matrix printers?

Line printers, dot matrix printers, & their ilk don't speak page description languages. Instead, they work on a simpler principle. They assume you give them data which they print in lines. They have simple control codes, such as

carriage return, line feed, form feed, & maybe a back-space. The important thing is that they do not speak a page description language. They assume the data you give htem is the data to print.

If you prefer, you might ocnsider their control codes to be a page description language. If someone told me that's the way it was, I wouldn't disagree, but notice that control codes make a very, very primitive & inflexible page description language when compared to PostScript.

I will not discuss line printers & dot matrx printers any more in this file.

Chapter 4

Printers & SNMP

4.1 What is SNMP?

SNMP stands for Simple Network Management Protocol, though I believe a more appropriate name would have been UCBDMP (Unnecessarily Complicated and Badly Described Manglement Protocol) because, where most RFCs are wonderfully understandable & useable, those of SNMP are not.

SNMP views network entities (those that respond to SNMP) as collections of key/value pairs. The keys are lengthy strings of integers, though the integers are sometimes expresed with alphabetic mnemonics.

The keys have a hierarchical structure, much like directories in a modern file system except that, where the separator between directory names is "/", "\", :", or possibly ">", the separator between the parts of an SNMP key is a dot (."). (By the way, this is not how the SNMP standard describes things.)

The basic operations of SNMP are:

- fetch the value for a specific key,
- fetch the values for a sequence of keys (performed using multiple, special fetch operations), &
- set the value for a specific key.

For more information about SNMP, see the relevant RFCs. There are a ton of them. One location of RFCs is the RFC Editor[1].

4.2 HP LaserJets

Hewlet-Packard's LaserJets set the standard.

attribute	note	oid
MAC addy		1.3.6.1.2.1.2.2.1.6.1

[1] http://www.rfc-editor.org/

4.3 Okidata

attribute	note	oid
okidata	oki-model	1.3.6.1.4.1.2001.1.3.1.1.11.1.0
oki-model	oki-model	1.3.6.1.4.1.2001.1.3.1.1.11.2.0
sys-model	sys-model	1.3.6.1.2.1.25.3.2.1.3.1
is-lying-oki	is-lying-oki	1.3.6.1.4.1.2001.1.3.1.1.11.1.0

4.3.1 Note: oki-model

1. If the *oki-model* is not the empty string, then...

   (a) If the *okidata* part is not the empty string, then it'll hold something like "OKIDATA", "Oki Data", "Oki", or something like that, so the full model name is what you actually got for *okidata* followed by the *oki-model*.

   Okay, so you are wondering how the *okidata* part could hold so many different values. You're asking why it can't be simple & always hold the same value. And I used to ask the same questions. Then I realized that with Okidata printers, everything is different. Well, not everything, but lotsof thins. Sometimes it's "Oki Data", other times it's "OKIDATA", still other times, it's empty. With Okis, each new model might ben an adventure.

2. Otherwise, the *model* part is the empty string, & all bets are off about the "Okidata" part. In fact, there's a good chance it's not an Oki at all.

4.3.2 Note: sys-model

When the *oki-model* OIDs don't give you want you want, try the *sys-model* OID.

Yet again, you're asking when to use which & why must it be so unpredictably different? And again I say, Because that's how the Okidata company works.

4.3.3 Note: is-lying-oki

4.4 PJL over SNMP

You can encode PJL commands in SNMP requests.

fixme: Document how to do tihs, why & when to do it.

You can at least do this for the model & the page count. Can you do it for other data? What models might require you to do this?

Appendix A

Other File Formats

- This document is available in multi-file HTML format at http://cybertiggyr.com/gene/hlj/.
- This document is available in Pointless Document Format (PDF) at http://cybertiggyr.com/gene/hlj/hlj.pdf.

Bibliography

[1] Adobe Systems Incorporated, editor. *PostScript Language Tutorial and Cookbook*. Adobe Systems Incorporated, 1985. ISBN 0-201-10179-3.

[2] Adobe Systems Incorporated, editor. *PostScript Language Reference*. Addison-Wesley, third edition, 1999. ISBN 0201379228.

[3] Eric S. Raymond, editor. *The Jargon File*. catb.org, 2003. http://www.catb.org/ esr/jargon/.

[4] Gene Michael Stover. Fun with postscript. *cybertiggyr.com/gene*, July 2005. http://cybertiggyr.com/gene/fps/.