Interview: Scott Pakin

David Walden

Scott Pakin has developed many LaTeX packages and other TeX-related tools.

Dave Walden, interviewer: Please tell me a bit about yourself.

Scott Pakin, interviewee: I'm 46 years old and have lived my whole life in the United States. I grew up in Chicago, Illinois (population: 2,700,000) and, after graduating high school, moved repeatedly to successively smaller cities and towns: Pittsburgh, Pennsylvania (population: 304,000) for my undergraduate degree at Carnegie Mellon University, then Champaign, Illinois (population: 232,000) for my Master's and PhD at the University of Illinois at Urbana-Champaign, and finally to Los Alamos, New Mexico (population: 18,000) to work at Los Alamos National Laboratory (LANL), where I'm still employed.

I knew from an early age I wanted to work with computers. I started programming in BASIC at age 9 on an obscure computer at my parents' company: an SDS 420 from Scientific Data Systems. (It used a 1 MHz 6502 processor and took 8-inch floppy disks, which contained maybe a hundred kilobytes of capacity.) When I was in high school, I wrote, in 8088 assembly language, a screen-dump utility called DumpHP that printed a screen of CGA graphics to an HP LaserJet printer. A small company named Orbit Enterprises licensed the code from me, incorporated it into their commercial LaserJet setup program, SetHP, and paid me royalties. Over the next few years, I made US$3000 in royalties — not bad for a teenager. I had no trouble deciding I wanted to get a bachelor's, master's, and eventually a doctoral degree in Computer Science. Along the way, I realized I especially enjoyed working with novel hardware and high-performance computers, and LANL has some of the world's fastest.

DW: Can you say something about the kinds of computing you do at LANL?

SP: At LANL, I've worked on a variety of research projects including tools for analyzing and improving the performance of supercomputers and the applications that run on them. I'm currently having great fun experimenting with a supercomputer we just bought from D-Wave Systems, Inc., that exploits quantum effects to solve a specific type of optimization problem. Think of TeX's paragraph-building algorithm, for example: It tries to find the best way to break paragraphs into lines to minimize the total penalty for awkward spacing. The research question I'm currently investigating is if it's possible to transform more-or-less ordinary looking computer programs into optimization problems suitable for running on a D-Wave system.

DW: How did you first come in contact with TeX?

SP: I began using the WordPerfect word processor (under DOS) to write documents. (WordStar was already losing popularity, and Microsoft Word hadn't yet caught on.) In my opinion, the best thing about WordPerfect was its "reveal codes" feature, which let one see the formatted document — calling it WYSIWYG would be overly generous — and the underlying markup (begin bold, end bold, begin italic, end italic, etc.) in a split-screen layout. Both were editable, but I really liked the precise control provided by the markup pane so I favored using that.

I hadn't heard about TeX until college, where a math-major friend who had recently learned LaTeX was excitedly talking about it. However, I didn't bother trying it out myself at the time. Once I began writing research papers in graduate school, I took the time to read through Lamport's book (first edition, of course) and learn LaTeX. Having been weaned on WordPerfect's "reveal codes" feature, I found LaTeX very natural to use.

Like most LaTeX newcomers, I managed to sloppily hack my way through whatever typesetting challenges I encountered. It wasn't until I started writing my dissertation that I decided to spend some effort on *really* learning how LaTeX works, how to use it more efficiently, and how to more precisely control its behavior. While doing so, I picked up TeX and LaTeX programming aspects and even wrote my first LaTeX package, `bytefield`, which I used in my dissertation.

DW: What was your dissertation topic?

SP: My PhD thesis considered processors distributed over a high-speed network working together to perform a computation fast. I presented an approach

for robustly synchronizing large numbers of such processors such that a few laggards don't necessarily slow everyone else down.

DW: You have a bunch of useful tools at CTAN[1]. Trying to grasp what is there, I can divide them into several categories: LaTeX packages; LaTeX metathings (e.g., `ctanify`, `dtxtut`, `bundledoc`); PostScript, EPS, and PDF related tools; tools for moving fonts into the TeX world (including some Metafont aspects), and combining TeX with other languages (Perl and Python). On your own website[2] you categorize things as packages, script, and documents. What motivated you to create all these different tools?

SP: Most of my LaTeX packages and programs were written to satisfy some typesetting need I had. Then, figuring that others might have the same need, I polished the code, documented it, and released it to CTAN.

It's always interesting to see which packages and programs have really caught on, and which quickly faded into obscurity. In fact, even *I* get surprised when I look over my list of CTAN contributions and find things I haven't used in years and barely even remember writing. From what I can tell, my `savetrees` package, which tries to squeeze a document into as few pages as possible, is wildly popular with researchers trying to stay within a mandated page limit for publication; and `attachfile`, which facilitates embedding arbitrary files within a document, and `hyperxmp`, which lets one include a large amount of metadata in a document, also seem to get a fair amount of attention. On the other hand, `spverbatim`, which enables verbatim text to wrap at spaces; `listliketab`, which typesets lists that arrange data in columns; and the `newcommand` script, which generates `\newcommand` templates for macros with complex juxtapositions of required and optional arguments, apparently get little or no use. Heck, I don't think *anyone* has ever used `dashrule`, which draws dashed horizontal lines.

DW: You may be too pessimistic about `dashrule`; it's recommended at `http://tex.stackexchange.com/a/125503` and is mentioned in many other places on that website.

DW: Both the Visual LaTeX FAQ and the Comprehensive LaTeX symbol list seem like they must have been enormous efforts. How have you gone about creating each of these?

SP: Indeed, the Visual LaTeX FAQ required quite a bit of effort to create, and the Comprehensive LaTeX Symbol List required substantially more. In both cases, one big challenge was to incorporate mutually conflicting elements in the same document. The symbol list, which tabulates a vast number of symbols that LaTeX documents can typeset, started with relatively few packages — base LaTeX, AMS, St Mary's Road, wasy — so it began being reasonably manageable. However, each new symbol package that gets added brings a new source of woe. Perhaps the biggest headache is that TeX has a hard-wired limit of 16 math alphabets. I typically have to access math fonts as if they were text fonts in order not to overflow that limit. Even worse, I've recently been encountering newer symbol packages that require LuaLaTeX or XeLaTeX, while some older packages break when using those TeX engines. Each new release of some symbol package seems to introduce a new conflict with some other package. As a result, the symbol list has become almost completely unmaintainable. I've begun work on a complete rewrite that should be robust to those issues, but that effort is slow-going and is still many years away from being usable.

DW: Please tell me your thoughts on the overall TeX infrastructure and whether you think it can be made better given practical limitations.

SP: TeX and LaTeX have a thriving infrastructure in terms of the sheer number of readily available LaTeX packages and the great strides being made in recent years enhancing the underlying TeX engine with improved support for system fonts and improved automation using Lua.

A practical limitation is getting new users to adopt the TeX ecosystem. Despite being only four years older than Microsoft Word, TeX has a far more "old-school" feel to it. Yes, TeX installation has improved over the years; and yes, GUIs do exist to simplify usage, obviate the need to learn control sequences, and provide word-processor-like synchronous editing. However, (La)TeX's lack of integration is a huge shortcoming relative to a word processor. If a user wants to typeset a table in a particular form, does he/she use an ordinary tabular environment or load one or more of the `array`, `bigtabular`, `booktabs`, `btable`, `calls`, `colortab`, `colortbl`, `ctable`, `dcolumn`, `easytable`, `hvdashln`, `longtable`, `ltablex`, `makecell`, `mdwtab`, `multirow`, `polytable`, `sltables`, `stabular`, `supertabular`, `tables`, `tabls`, `tabu`, `tabularborder`, `tabularew`, `tabularht`, `tabularkv`, `tabularx`, `tabulary`, `threeparttable`, `threeparttablex`, or `xtab` packages? Even worse, many packages conflict with each other either explicitly (giving an error message) or implicitly (screwing up some unrelated aspect of the document in some hard-to-diagnose manner). Worse

still, the set of conflicts can change from version to version of any given package.

Another example of (LA)TEX's lack of integration relative to a word processor is that a word-processing document is stored in a single file that can easily be transmitted to colleagues. My `bundledoc` script helps with this on the LATEX side by bundling together all the separate document files, style files, class files, graphics files, etc. into a single `.tar` or `.zip` file, but usage is still a bit clunkier than what an integrated tool can provide.

Word processors have been improving their typesetting quality, support for mathematics, support for international scripts, logical structure, and other features that have traditionally lain in TEX's wheelhouse. For most users, word processors are good enough tools for the jobs they have. I think the wrong approach is to try to turn (LA)TEX into a word processor. It lacks the foothold of, say, Microsoft Word and is unlikely ever to become a dominant form of document interchange. Instead, (LA)TEX infrastructure enhancements should focus on the system's core strengths: ease of making global, structural changes to an entire document; ease of automation; and ready and convenient support for a variety of specialized typesetting requirements in areas such as linguistics, mathematics, and natural sciences.

DW: Given you've built various PostScript, EPS, and PDF tools (such as `purifyeps`), do you have thoughts on what might practically be done with (LA)TEX to make them more suitable for integrating with PDF et al.?

SP: I guess I don't have any grand vision for better integration of (LA)TEX with the PDF world. That said, native support for PDF/A-1a generation and fully tagged PDF would be nice. The former guarantees a high degree of portability, and the latter facilitates reflowing text on a tablet and improves mechanical reading of a document to the vision-impaired.

DW: At the 2014 TUG annual conference in Portland, Mertz, Slough and Van Cleave presented a paper[3] that included a significant discussion of your `bytefield` package. Also, Mertz and Slough previously presented a lengthy discussion[4] of your PerlTEX system. I am interested in how you feel about other people describing your work and whether they interacted with you as they wrote their papers.

SP: I'm always eager for people to use my tools. It's wonderful to know that I helped someone get the typesetting they were looking for or automate some tedious task.

I was not contacted by the authors of the papers you cite above, but that's probably a good sign; it says the authors were able to get `bytefield` and PerlTEX to work without extra help. For PerlTEX, which lets users write LATEX macros in Perl, that's especially encouraging. PerlTEX was extremely challenging to develop and is therefore likely to be a lot more fragile than a typical LATEX tool. It requires a lot of TEX trickery to process what could be considered syntactically incorrect TEX but syntactically correct Perl from within TEX, and it takes a Computer Science-y distributed-systems-style protocol to implement correct, two-way communication between TEX and a Perl wrapper script given TEX's limited ability to communicate with the outside world in a safe (i.e., not-`\write18`) and portable manner. It's great that PerlTEX works fine for Mertz and Slough and that they were able to perform some interesting and creative tasks with it.

DW: Do you still see a role for PerlTEX with LuaTEX now available?

SP: Not so much. LuaTEX deeply integrates Lua with the TEX engine while PerlTEX is more loosely coupled. Consequently, there are things LuaTEX can do that PerlTEX can't (e.g., directly manipulating some of TEX's internal representations). On the other hand, I find PerlTEX's `\perlnewcommand` and `\perlnewenvironment` macros very convenient. Perhaps I should write a package that provides the analogous `\luanewcommand` and `\luanewenvironment` macros ....

DW: You've also developed lots of other tools, such as those listed on your personal website (readers: see `http://www.pakin.org/~scott/`). Perhaps you also do not hesitate to build a new tool in the course of accomplishing your work at LANL[5]. Can you speak about tradeoff between (a) just doing what you have to do to accomplish some primary task, and (b) first building a tool to help you with the primary task and then applying it to accomplish the task?

SP: I write lots of tools, and I always learn something new when I do. It's always a good idea, though, to perform a task manually the first few times to determine what aspects are suitable for generalization and automation — and to convince yourself that the task is in fact something that gets performed sufficiently often as to warrant building a tool for it. I suppose the following are a good set of questions a tool-builder might ask himself before embarking on developing a new tool or, in the context of this discussion, a new LATEX package:

- Is the task sufficiently common as to warrant building a tool for it?

- Is the task sufficiently complex for users to be willing to install and learn a new tool rather than perform the task manually?
- Is the task sufficiently general for a tool to perform it without having to be so parameterized that it becomes almost as difficult to learn and use as it is to perform the task manually?

DW: Were any of these tools particularly more fun to work on?

SP: It's hard to pick a single, most fun piece of LaTeX development. PerlTeX is the most sophisticated LaTeX-related tool I've ever created, and it was exciting when I finally got that to work. My three standalone documents — The Comprehensive LaTeX Symbol List, The Visual LaTeX FAQ, and How to Package Your LaTeX Package — all required a fair amount of thought to produce, and I learned quite a bit from each one. I guess the common thread is that tools, packages, and documents that were intellectually challenging to develop are more rewarding than those that required only straightforward coding.

DW: Thank you for taking the time to participate in this interview. You are working on a lot of fascinating things.

[Interview completed 2017-02-05]

Links

[1] https://www.ctan.org/author/pakin
[2] http://www.pakin.org/~scott/latex-stuff.html
[3] https://www.tug.org/TUGboat/tb35-2/tb110mertz.pdf
[4] https://www.tug.org/TUGboat/tb28-3/tb90mertz.pdf
[5] https://ccsweb.lanl.gov/~pakin/

◇ David Walden
http://tug.org/interviews