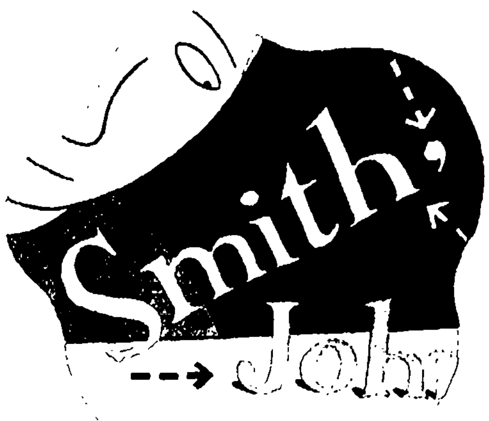
Addresses, Part II

by Jeffreys Copeland and Haemer

Welcome to the fourth in our series about common office problems and our solutions to them. Our plan is to share some of the tools we've developed over the years to keep us orga-



nized (though our families would scoff at the idea that they actually work) and to explain the design decisions that went into them.

Last Month's Problem

Last month, we started to develop a scheme for handling your address book with some simple shell scripts. We provided two alternative approaches. In the first, we used a flat ASCII file with paragraphs separated by blank lines as our electronic address book, like the following:

```
Jeff Copeland
QMS GLIF Group
4730 Walnut Ave., #202
Boulder, CO 80301
#w: 303-443-7227
#f: 303-443-7107
% jeff@rd.qms.com
% copeland@alumni.caltech.edu

Jeffrey S Haemer
Canary Software
960 Ithaca Dr
Boulder, CO 80303
#w: 303-494-0924
#f: 303-494-7514
%jsh@canary.com
>xmas
```

Jeffrey Copeland (copeland@alumni.caltech.edu) *is a member of the technical staff at QMS's languages group, in Boulder, CO. His recent adventures include internationalizing a large sales and manufacturing system and providing software services to the administrators of the 1993 and 1994 Hugo awards. His research interests include internationalization, typesetting, cats and children.*

Jeffrey S. Haemer (jsh@canary.com) *is an independent consultant based in Boulder, CO. He works, writes and speaks on the interrelated topics of open systems, standards, software portability and porting and internationalization. Dr. Haemer has been a featured speaker at Usenix, UniForum and Expo Kuwait.*

We built a pair of utilities: crunch, to collapse paragraphs into single lines so we can grep and sort files of them; and uncrunch, which is the inverse of crunch.

In the second scheme, we adapted the standard utilities from the refer suite to our address book application.

Last month, as we often do, we also left you with a small problem: Given paragraphs with names like "Gillian Haemer," "James Joseph Schwarzin-Copeland," "Ludwig von Beethoven," "Albert d'Andrea" and "Sean O'Malley," write a program to invert them (to "Haemer, Gillian," for example) so we can sort them. Let's begin with the simple case of lines alone and then generalize to paragraphs like those above. Our first cut looks something like this:

```
#! /bin/sh
# reverse first and last names:
# the regular expression for last
# name will catch names like O'Malley,
# vonBeethoven (but not von Beethoven)
# and Smythe-Hamilton
sed "s/\ (.*\) \([-a-zA-Z\´]*\) \$/\2, \1/"
```

As you can see, this looks for two strings anchored at the end of the line. The second cut is a string of characters consisting solely of alphabet letters, apostrophes and dashes at the end of a line preceded by a blank (let's call it the surname). It then inverts the order of the two strings, separating them by a comma. To allow us to add a name like "von Beethoven," we need to do some fancy footwork to get around the space in the middle:

```
#! /bin/sh
sed -e "s/ \ ([a-z]*\) / \1_/" \
    -e "s/ \ (.*\) \(-_a-zA-Z´]*\) \$/\2, /" \
    -e "s/_/ /"
```

Notice that, aside from skipping the comments, we've replaced the space between any solely lowercase connective with an underbar. We then add an underbar to the allowable character set for the surname. We finish by replacing underbars with blanks. By doing this procedure in three steps, the first and last substitutions don't happen if there isn't a prefix like "von" in the surname. However, our scheme fails if the surname has multiple connectives, such as "de la Guardia" or a capitalized connective, such as "Von Braun."

(Exercise for the reader: How would you handle names with extra qualifiers at the end, like Martin Luther King, Jr. or Richard P. Feynman, Ph.D.?)

The last step is to make this process work for our paragraph scheme. We obviously can't just run it on the raw data, or we'd get lines in our addresses like "Software, Canary." If we remember that our crunch command from last month uses a backslash as a separator, we can incorporate that into our sed script, like so:

```
#! /bin/sh
crunch |
sed -e "s/ \ ([a-z]*\) / \1_/" \
    -e "s/\ ([^\\]*\) \([-_a-zA-Z\´]*\)\\\\/\2, \1\\\\/" \
    -e "s/_/ /" |
uncrunch
```

We collect the forenames with an expression that looks for repeated non-backslashes. This prevents our getting the last word on a later line of the original paragraph, such as "Software."

The net effect is that we can execute a command like

```
crunch addr | revnames | sort -fb |uncrunch
```

to sort the address book.

(Here's another exercise for the Perl-literate reader: Render the above sed scripts into Perl. We'll show you how in the next installment. Perl turns out to be amazingly powerful for text processing. One of us just completed versions of addbib and sortbib replacements for his BSD-based system entirely in Perl.)

User Interface

There is an old joke in the BSD fortune file about Ken Thompson's car. It doesn't have any instruments, save for a giant question mark that lights up on the dashboard when the driver makes a mistake. "The experienced driver," Thompson is quoted as saying, "will usually know what's wrong."

Well, we feel the same way about user interfaces sometimes: Less really is more. In most formal database management systems, data goes in through a fancy program interface, is encoded in some fashion on the disk and then needs some other kind of special-purpose program to get it out again. The same applies to many graphical user interface programs, like word processors in the Windows and Macintosh universes–have you ever tried to extract the text from a Windows help file?

However, the UNIX philosophy obviates all this special-purpose fiddling about. We have text files in some format, in simple ASCII codes, and use filters to extract the data from them. We find this much simpler than having to write 1,500 lines of code for an interface using routines like Open_DOS_Handle_Interface_for_New_Window() –or was that Open_DOS_Handle_Interface_for_ Old_Window()? (Yes, gentle reader, not only did we curmudgeons walk five miles to school in the snow, uphill both ways, we also used to program using punched cards.)

This brings us back to our use of refer to handle our address book. If you'll notice, the data comes out of sortbib in roughly the same form as it went in. If I use lookbib and query for "Haemer," I get:

```
%F Jeffrey S
%L Haemer
%B Canary Software
%S 960 Ithaca Dr
%C Boulder
%Z CO 80303
%W 303-494-0924
%F 303-494-7514
%E jsh@canary.com
%T xmas
```

Because of this "what goes in comes out" approach, we can use a command line like sortbib -sLF to sort the names in our address book. We can pipe the output of this into roffbib with a version of the -mbib macros modified to print out an address book rather than a bibliography. We did the same thing to transform letters into envelopes. We wrote a version of the letter macros that only printed the writer's and recipient's addresses. We'll leave modifying the -mbib macros as an exercise.

Printing the Address Book

Nonetheless, we can print the non-refer version of the address book with a fairly simple script. We get the address book files on the command line, with some defaults.

```
#! /bin/sh
# address book printing --- sometimes it's just
# a pain to fit the computer in your Filofax

# names of the files to print
[ -n "$*" ] && files="$*" || files="addrz"

lines=43       # default page length

x=
for i in $files
do
        [ -n "$x" ]    &&      echo    .bp
        precol -l$lines $i
        echo .EE
        x=zz
done | \
tps -D -F HelvNarrow addr.mac -
```

Several things to notice: First, we've used a new filter, precol, which adds blank lines to the bottom of "pages" to even them out and not break paragraphs across pages. Why don't we use troff's ne directive to cause a page break if we have too few lines on the page? Because we don't want to preprocess the whole file, counting the lines of each paragraph. (We will explore precol at a later time. It turns out to be a useful tool for any application where the data is blank-delimited paragraphs.)

Next, we use the EE macro at the end of each file. We pipe the lot into a troff wrapper that allows us to choose duplexed output (-D), and the font family (Helvetica Narrow). We use a macro package built for just this purpose. (We could have used the standard -mm with a mass of parameters instead of writing a new package, but this gives us better control over margins and page size.)

As for what addr.mac contains:

```
.\"    address book macros
.\"    ------------------------
.\"    parameters and setup
.nr    NC 2              \" #col
.nr    nc 0 1
.nr    TM .325i          \" top margin
.ps    8
.vs    10p
.ll    3.25i
.lt    3.25i
.pl    6.75i
.nf
.wh    \n (TMu+\n(NLv }T
.\"    --macros
.de    OP                \" offset on page
.ife     .po 8.5i-\\n(.lu-lc
.ifo     .po 1.4c
..
.de    }T                \" bottom of page trap
.ie    \\n+(nc>=\\n(NC \{\
.bp
.nr    nc 0 1
.OP    \}
.el    .po + (\\n(.lu/\\n(NCu)
.sp    |\\n(TMu
..
.de    EE                \"end of file macro
.sp    \\n(.tu-1.lv
.af    mo i
.OP
.tl    '''Printed: \\n(dy.\\n(mo.\\n(yr'
.ifo   \{ .bp
.sp    1i
.ce    2
This page intentionally
left blank
.\}
..
.sp     |\n(TMu
.OP
.ns
```

We won't go into excruciating detail about these macros. They are set up in default to produce pages to be trimmed into the common 6¾- by 3¾-inch notebook. Notice that when we end a section on a recto, we print the self-contradictory statement `this page intentionally left blank` on the verso–we are, after all, computer nerds.

Useful Extractions

Another interesting use for our address book is printing envelopes. Normally we'd just generate an envelope from a letter (see "Envelopes," May 1995, Page 35).

In some cases, though, we want to print an envelope without a letter. Bank deposits come to mind: I want to print an envelope with the bank's address but don't want to write a letter or even set up a file with a dummy letter in it just to print the envelope.

Let's say I've got an entry in my address book like:

Another interesting application for our address book is printing envelopes... In some cases, we want to print an envelope without a letter.

```
First National Bank of Duckburg
42 Scrooge McDuck Blvd
Duckburg, Disneyland
```

I want to turn this into a temporary file like:

```
.WA
2945 Wilderness Place
Boulder, CO 30303
.WE
.IA
First National Bank of Duckburg
42 Scrooge McDuck Blvd
Duckburg, Disneyland
.IE
```

and then feed it to our old doenv script.

Assume we invoke the envelope printer with the same arguments as our `call` program to look up names and addresses. Then we can simply wrap the names inside of `IA/IE` pairs, adding a return address.

```
#! /bin/sh
# getenv --- get addresses out of our
#  address book and print envelopes
for i in $*
do
        call $i | egrep -v "[#\>]"
done |
(echo ENVB;
sed -e '$d' -e 's/^$/.IE\
ENVB/'
echo .IE) | \
sed -e 's/ENVB/.WA\
2945 Wilderness Place\
Boulder, CO 80302\
.WE\
.IA/' >/tmp/$$.env
```

Actually, we've done some additional things here. First, we remove the lines from the address book entries for phone numbers and email addresses. We massage the output of this lookup by prepending a place-holder for "give us a return address, begin a letter." We append the "finish this letter" macro. And in between, we replace every blank line with "finish this letter...begin a letter." We invoke a second sed script on the whole output that expands our "begin an envelope" shorthand. We've put the whole result into a temporary file.

The next step is to invoke the doenv script we wrote earlier. Obviously, we could just pipe the output of our sed script so far to doenv.

But what if we have a script for printing package labels from letter text? This would be a logical script to extract the addresses and print the labels, except that at this point we'd invoke `dolabel` instead of doenv. So, our last bit in this script is not `doenv /tmp/$$.env` but rather

```
pgm=`echo $0 | sed s/get/do/`
eval $pgm /tmp/$$.env
rm /tmp/$$.env
```

Summary

We've looked at printed address books, and methods of manipulating the data in the electronic version. We still owe you an explanation of the `precol` filter. We didn't look at other methods of extracting data from our address book in an organized fashion, such as "everyone who lives in Southern California." We will look at this problem next time, along with some others. Until then, enjoy your summer. ▲