Addresses, Part I

by Jeffreys Copeland and Haemer

With the coming of summer, a young consultant's fancy turns to...updating his address book.

This is the third in our series about common office problems and our solutions to them. We plan to share some tools that we've developed over the years and some new tools that we've developed specifically for this series. Our plan is not only to present the tools, but also to explain some of the design decisions that went into them–though this month, because we're showing a tool a minute, we'll skimp a little on design principles.

The Problem

You've got a Rolodex, an address book, a business card file and a bunch of preprogrammed buttons on your telephone, and you still can't find a colleague's phone number–or remember to call your mother on her birthday. We'll solve the first problem this month, and in the meantime, Mom will forgive you.

The classic tool for finding things

is, of course, grep. You could build a file, phonelist, containing one-liners such as:

```
Jeff Haemer office 303-499-8924
Jeff Copeland office 303-443-7227
Canary Software fax 303-494-0924
Ian home 011-852-528-6605
Softbank 011-03-818-7531
```

and then find all the facsimile numbers with a command like `grep fax phonelist`. Similarly, you could get Jeff's phone number with `grep Jeff`

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.*

```
phonelist.
```

So far, so good, but what about addresses? Often, I need to know more than Canary Software's phone number. I want to know its fax number too, and that its address is 960 Ithaca Drive, Boulder, CO 80303. I want to be able to access that information if I ask for either Canary's phone number or Jeff Haemer's. So I really want a file that looks like this:

```
Debbie & Ian Copeland
27D Seabird Lane
Discovery Bay
Hong Kong BCC
#w: +852-528-6605
#f: +852-529-6672

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 Drive
Boulder CO 80303
#w: 303-494-0924
#f: 303-494-7514
% jsh@canary.com
>xmas

Guy H Lillian
1101 Franklin
Jefferson LA 70053
#h: 504-827-9285
>xmas

Dr T A Dolotta
Softbank Research Institute
Wako Building
2-31-25 Yushima
Bunkyo-ku
Tokyo 113 JAPAN
#w: 011-03-818-7531
#f: 011-03-818-7534
#h: 213-453-8649
```

Notice that in this paragraph-based scheme, we use some rudimentary tags to indicate types of data. We have a block of lines with a name and address, with a telephone number tagged `#w:` (or for a home number, `#h:`) or a fax number tagged `#f:`. Similarly, we have email addresses tagged with a percent sign, and we use text tags preceded by >, such as >xmas for folks on our Christmas card list.

With this scheme, we actually have several separate problems. The first is to extract data from this list. There are a number of ways we can do this, most obviously by name (let's say I want to call Ted Dolotta), and then by criteria such as location (in the event I'll be in Washington, D.C., next week, and want a list of everyone I know in the area) or tag (in case I need an email address book or want to print my Christmas card list).

The second problem is to format that data appropriately. In the simplest case, I just want to look at the equivalent of the Rolodex card on my screen. In a more complicated case, I want to format the data I've extracted onto pages that fit in my pocket calendar. An additional problem we may want to attack is massaging the extracted data–for example, presenting the list in alphabetical order.

Let's try to solve the extraction problem first and leave the formatting problem in its full glory for later. For a first attempt, we can just say: `grep -i jeff phonelist`. Unfortunately, we get:

```
Jeff Copeland
% jeff@rd.qms.com
Jeff Haemer
```

Where are the phone numbers? We need a variant of grep that allows us to extract a blank-delimited paragraph.

Luckily for us, Steve Zucker at Interactive Systems Corp. spent a Saturday afternoon developing this extension to grep more than a decade ago. (It was Steve who, in frustration during an interminable meeting, uttered the immortal line: "No, we're not going to port that program to the new system. We didn't design it before we wrote it the last time.")

Unfortunately, Steve's `grep -p` isn't part of the standard, and Interactive dropped it when it began supporting standard versions of systems. But fortunately for AIX users, it is still available on RS/6000s as an extension. But we're trying to develop general solutions based on the standards here, so bear with us.

There are two alternatives to our paragraph grep problem. The first is to cause the paragraphs to become lines somehow. The second is to remember that awk allows us to arbitrarily redefine the record separator. Let's briefly explore both possibilities.

Some Solutions

First, we can make paragraphs into single lines with a program to collapse them. For example:

```
#! /bin/sh
# Crunch blocks of multiple lines
# separated by a blank line onto a
# single line, for grepping, etc

awk '/./ { printf "%s\", $0 }
/^$/ { printf "\0 }
END { printf "\0 } ' $*
exit 0
```

To reverse the process, we use:

```
#! /bin/sh
# Inverse of crunch:
# take a single line composed
# from the text from a block
# of blank-delimited lines,
# and restore it to a block
# of separate lines.

awk ' { gsub("\\\\", "\n", $0);
     printf "%s", $0 }' $*
exit 0
```

So, as a result, if we have our phone lists in $PERS/addr/addr?, our first-cut script to look up addresses and phone numbers is:

```
#! /bin/sh
crunch $PERS/addr/addr? | \
    /bin/grep -i "$1" | uncrunch
```

(Some exercises for our readers: What happens to files containing backslashes when processed by crunch and uncrunch? Can you fix this? Can you generalize them to use an arbitrary line separator? How does an arbitrary line separator affect the other programs we're developing in this column?)

Alternately, as we mentioned, we can use our old friend awk if we remember that we can redefine the record separator to a blank line with

```
BEGIN { RS = "" }
```

so we can use a script like

```
#! /bin/sh
awk "
BEGIN { RS =\"\" }
/$1/ { print; printf \"\\n\" }" $PERS/addr/addr?
```

There is still a small problem, though, because if we're looking for "Jeffs," the output we get is

```
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 Drive
Boulder CO 80303
#w: 303-494-0924
#f: 303-494-7514
%jsh@canary.com
>xmas

Guy H Lillian III
1101 Franklin
Jefferson LA 70053
#h: 504-827-9285
>xmas
```

Where did Guy's address come from? He lives in the city of *Jeff*erson. We've always just lived with this problem, but how would you solve it? (And similar ones such as getting all your contacts in Maryland when you are looking for a Medical Doctor.)

Over a cup of coffee one evening in the Haemer living room, we realized that even though we've been using ad hoc tools like this for years, there is a standard UNIX too to perform these tasks.

Inverted Indexes

Mike Lesk built refer around the idea of inverted indexes. His intention was to have a useful bibliographic database tool to insert references into troff source. We can use refer instead to build a tagged database. The addbib command conveniently allows us to enter data to an arbitrary set of tags, such as

```
First          %F
Last           %L
Company        %B
Street         %S
City           %C
State ZIP      %Z
Home #         %H
Work #         %W
FAX            %F
e-mail         %E
tags           %T
```

A command like addbib -a -p addr.prompts addr creates an address database with a dialog like

```
canary> addbib -a -p zz.prompt addr.bib
Instructions? n

First          Douglas
Last           Morgan, MD
Company        Kaiser Permanente
Street         Broadway & North
City           Boulder
State ZIP
Home #
Work #         303-440-0884
FAX
e-mail
```

```
tags

Continue? n
```

Of course, we can just edit the database with vi, too. This means that we can look up phone numbers with a tool based on lookbib:

```
#! /bin/sh
## use refer to look up a
## phone number and address
lookbib addr.bib 2>/dev/null <<EOF
$1
EOF
```

It also means that we can sort the database. For example, sortbib -sLF will sort by last and first names–we'd need an invert-name command (to change "Gillian Haemer" into "Haemer, Gillian") so we could do this with our free-format database:

```
crunch | invert-name | sort | uncrunch
```

(Exercise for the reader: Write an invert-name that properly handles Gillian Haemer, James Joseph Schwarzin-Copeland, Ludwig von Beethoven, Albert d'Andrea and Sean O'Malley. We'll show our solution next month.)

The disadvantage of the refer approach is that we need to define our fields carefully beforehand. We can end up forcing data into our mold in a case like this:

```
%F Dr T A
%L Dolotta
%B Softbank Research Institute
%S Wako Building, 2-31-25 Yushima
%C Bunkyo-ku
%Z Tokyo 113 JAPAN
%W 011-03-818-7531
%F 011-03-818-7534
%H 213-453-8649
```

Furthermore, I can't look this entry up by the keyword japan or by the phone prefix 011-03 because lookbib doesn't scan those fields. Also, unlike our free-format database, we have a variety of tags directly from lookbib, which we need to edit out before we can drop the information into a letter or other file.

When we continue our series next month, we'll demonstrate this with the free-format database, rather than the refer-based one.

Next Time

This month, we've attacked the problem of setting up an address database and looking up specific entries in it. We've shown you both a free-format data file approach and one based on Mike Lesk's refer program.

Next month, we'll show you the rest of the tools you'll need to manage your address book, such as looking up entries by class (this is the "I'm visiting Washington..." problem we talked about earlier). We'll also show some tools for formatting those entries. There are two kinds of problems here: Print me an envelope for Ted Dolotta, and print my entire address book on paper so I can throw it in my briefcase.

Thanks to our friend Rob Tulloh whose knowledge of AIX internals is much better than ours and who has provided some useful information for this series. ▲

Reader Feedback

To help *RS/Magazine* serve you better, please take a few minutes to close the feedback loop by circling the appropriate numbers on the Reader Service card located elsewhere in this magazine. Rate the following column and feature topics in this issue.

	INTEREST LEVEL		
Features:	High	Medium	Low
Matching the Right Piece to the Puzzle	170	171	172
A Buyers Guide to Printers	173	174	175
Juggling Distributed Jobs with DSMIT	176	177	178
Columns:			
Q&AIX–Shared Libraries	179	180	181
Systems Wrangler–Fencebuilding 101	182	183	184
Datagrams–Is the Internet All Male?	185	186	187
AIXtensions–Managing Data Madness	188	189	190
Work–Addresses, Part I	191	192	193