LaTeX And The Personal Database

Bernice Sacks Lipkin
9913 Belhaven Road
Bethesda MD 20817
USA
bslipkin@erols.com

Abstract

Ignoring fixed-size and coded field formats, text databases can be viewed as either ASCII-delimited or ID-prepended. ASCII-delimited databases reserve a particular symbol as record delimiter, and another symbol as field delimiter. ID-prepended databases mark the start of each new field with stylized text unique to that field type. *Personal* means that the record can take any form the database owner desires, from the rigidly-structured, where the informational fields are in the same order in each and every record in the database, to the totally unstructured records of ordinary documents.

TADS is a set of integrated programs that manipulate the text in a personal database. Using TADS, field-specific LaTeX instructions can be made an intrinsic part of an ASCII-delimited database from its inception. To create a bibliographic data base and incorporate items from the database into manuscript text requires that the writer do two tasks:

1. Using TADS, the writer creates a customized data entry program that will prompt for keyboard input to create the field order he wants; e.g., *author, title, journal, etc.*. The data entry program writes a skeleton LaTeX file, complete with skeleton commands, one per field. These macro names, which also prepend the fields in each database record, define font size and shape.

2. In writing a document for publication, the writer alludes to citations in the document, using whatever phrases he can recall. The allusions are written within brackets; for example, *Smith et al <smith*199?*tadpole*> suggested...* Under a script that *stacks* TADS program modules with the appropriate options, these allusions are extracted from the document, and serve as wildcard match words to pull out records in the derivative database. Records are automatically sorted by number or by Name-Year and the numbers (or Name-Year) substituted for the allusions in the document. Fields can be rearranged or omitted in the bibliography; and the accompanying LaTeX field macros can be redefined for the font and style requirements of the particular journal.

Introduction

This paper presents some of the ideas that are explicated more fully in a monograph in preparation on human-aided computer manipulation of bibliographic databases.

TADS[1] is a set of gofer utilities for text processing. It has no artificial intelligence, no understanding of the meaningfulness of the text that it finds and manipulates. At the other extreme, in contrast to data mining, it doesn't pull out patterns statistically. It does recognize features of the text. It can pick out vowels or digits or all the letters of the alphabet or all the alphanumerics. It knows the start of a field from the rest of the field and where fields are located on disk. It can match words exactly or find a word in the field or do wildcard matching. With these simple skills, it does text substitution

1. TADS is an acronym for Text Analysis, Description and Synthesis. It is the latest reincarnation of a set of text manipulation and string processing programs. Variations of many of these programs were first written in the 1970s in Sail; the system was called *MaText*. A later version called *TXT* was written for the Microsoft C compiler and a DOS-Windows environment. It is documented in *String Processing and Text Manipulation in C*, which came with a copy of the source code for many of the functions on floppy. The book is currently being revised for a Linux-Unix environment. The programs now work with any size record.

and sorts records alphabetically or by class. There's even a job called *lazyboy sort*, where the machine figures out subclasses by examining the text. It compresses records vertically and links files horizontally. It makes statistical tables by tallying frequencies of linked words. It has no preconceived ideas about how to terminate a record or a field; you tell it the record delimiter, and if the record has more than one field, the field delimiter. It does insist that you reserve these marks — usually punctuation marks — as delimiters; that is, you can't use them in the body of the text.

TADS works with the text you give it, whatever it is. It works with your personal database, one that you design. If the input is a flat relational database — where the number of fields in each record is the same, the types of information in the sequence of fields are the same and fields are not subdivided — it can treat the fields as columns, and do all the *find* and *retrieve* tasks we expect from a search engine operating on a highly-structured simple database. But it is just as happy if it is directed to operate on a semi-structured database, where the first fields are predictable, and the later fields are unstructured — clinical notes for example. It can handle hierarchical fields, which include subfields or even subsubfields, a format that truly keeps data that belong together together. TADS treats ordinary manuscripts as records, if you declare the period as record delimiter. To partition a sentence, declare the comma as field delimiter. Obviously, not as many operations can be done on unstructured fields. But you can, for example, treat an entire book as a single record and extract all the test phrase that are bracketted by <> or [] or () or whatever.

In Figure 1, the first example is a flat database record, one where each field contains a single item of information. The second keeps all the data on jobs in one field. It is an example of a hierarchical database, where the field is subdivided into subfields, and the subfields are partitioned into subsubfields. The number of subfields usually vary from record to record. Subsubfields are generally rigidly structured because they are usually designed to provide quantitative results dependent on information that is ordered in time or in some other dimension.

Adding an unstructured Notes field to the first record would not affect its processing but would contribute to its functionality. It could act as a reminder of the times the consultant was used. The field could also be made private for evaluations and assessments by filtering the Notes field out before the file could be examined publically.

TADS searches are fast, but not as fast as *google* or *dogpile*. Much of what it does is done by commercial database managers. It does, however, have some nice features. It can handle any size record, any size field. It operates just on the fields you want processed. Any and all databases can be read and modified by you whenever you wish. (Of course, if you change a file that is searched by way of a TADS-created index, you will have to — or TADS will have to — redo the index.) The original database is always readonly, but TADS can send records that were modified to one file, records that were unaffected to another, eventually creating a genealogical *tree* that, properly manipulated, does the equivalent of AND, OR and NOT booleans.

When the programs were applied to databases with relatively simple structure — lists of bibliographic citations — I ran into a practical problem that was not large and interesting but small and annoying: the impossibility of picking a font format that would need no revision. For efficiency, markup instructions are usually embedded directly into the database text from the start. This can, however, produce problems downstream. What if one journal wants Volume numbers bolded, another wants them italicized, the next slanted. Redoing font instructions can be laborious, especially if you need to change the markup instruction in only one or two specific fields in every record, whenever you must use a different font. Alternatively, it is possible to maintain a database in unformatted text, adding exact instructions as needed. But this suffers from the same need to revamp much of the text.

More recently, I've started working with the ID-prepended format. An ideal example is an item from a MEDLINE download, such as the one shown in Figure 2. (I've omitted the Abstract field.) Keys to the article are listed individually in the Mesh Heading (MH) fields. The download format is stylized. Field names are two-letter, followed by two spaces, a hyphen and a space. What is a single hierarchical field in an ASCII-delimited record is split into individual fields. Field order is always the same. The Universal Identifier (UI) is always first. It is always followed by the author(s), the title, the language, and so forth.

Look at the figure. It is obvious that if you prepend a backslash to the field ID, substitute a left brace for the " - " and add a right brace at the end of the field, you have a LaTeX macro command with its text argument, a different macro command for each type of field. Naturally, you must define this new command, flesh out the font size and shape. By defining a macro for each type of field in the preamble, you can control the print characteristics

Flat field. The record delimiter is **!**. The field delimiter is /. They can not be used in the body of the record. Notice that the end of the final field is not a diphthong; i.e., it terminates with the record delimiter, not the field and record delimiters.

```
ConsultantFile0027/1994/
Smith, John M./electrical
engineering/Ph.D., U
Calif./99 First Street,
Lakeview 11111 WI!
```

Hierarchical field. The field delimiter is /, the subfield delimiter is % and the subsubfield delimiter is **$**. Two fields are shown: one simple, one hierarchical. John Smith may have held any number of positions, but each position subfield has a fixed number of subsubfields, three in this example — title, university and starting date.

```
John M. Smith, Ph.D./
Ph.D.$U. of Calif.$1957%
PostDoc$Yale$1959%
Asst Prof$Yale$1963%
Assoc Prof$Wisconsin$1967%
Prof$Wisconsin$1971/
```

Figure 1: Two types of database fields.

of the individual fields. This doesn't solve all the problems of modifying print appearance on the fly. But it helps.

The notion of tagging a record with a combined command name and field identifier when it is added to the database can be applied to ASCII-delimited records. The format I'm currently exploring is transitional; it has characteristics of both the ASCII-delimited and the ID-prepended formats. It still relies on a record delimiter, the ~, and field delimiters, the /, to isolate and partition the record. This is an example after it was keyed in under a data entry program designed specifically for these field types and the fields rearranged.

```
\ACNUM{DemoFile01}/\NAMEYEAR{Eisthen, 1992}/
\AUTHOR{Eisthen, H.L.}/\YEAR{1992}/\TITLE{Phylo-
geny of the Vomeronasal System and of Receptor
Cell Types in the Olfactory and Vomeronasal Epi-
thelia of Vertebrates}/\PAGES{1-21}/\JOURNAL{Mi-
crosc. Res. Tech.}/\VOLUME{23}/\ISSUE{1}/ / /
/ / /\NOTES{93004928}~
```

A database is stored in ASCII with prepended IDs, which are actually LaTeX macro command names. The end of a field in the current version is redundant: it has both a right bracket that we need for LaTeX anyways plus a field delimiter. In some files, I've eliminated the field delimiter altogether, using the } both for LaTeX syntax and field delimiter. But I won't do this in a general way until I've convinced myself that there's no interference with TADS in general, or at least in the major ramifications and combinatorics of using its modules in a sequence to get a particular result. The current format clearly does not interfere with the programming jobs that turn a database record festooned with tags and extraneous information into a well-behaved reference suitable for publication.

At this point, you are probably thinking BibTeX. BibTeX is indigenous to LaTeX. It has a multitude of formats, is easy to use and does much of its work transparently. TADS was not designed as a bibliographic database manager. It has little terminology specific to bibliography. In fact, it has little terminology. Labeling fields is a major innovation. But it remains a set of general programs, each with multiple options, that leaves it up to you to work out the correct sequence of programs and the appropriate options to do a particular task. Once a database is created and a map for obtaining the desired result is drawn — e.g., run *dork* task 3 with these options, then run *addtext* task 4 with these options, and so forth — it is simple enough to run the job from a script.

One such job is integrating a canonical bibliography and manuscript for a particular journal. To run the programs that do this, you first need to do two things:

Acquire a database.
To use TADS, you must add at least a record delimiter or, better still, a record delimiter and field delimiters, to each record in the file. That's it. TADS will extract a list of citations from a database, alphabetize it, number it, and substitute the numbers in the manuscript. But there are advantages to building in LaTeX macro names from the start. This paper describes a program that writes a data entry program to supervise the creation of a database from text

```
UI  - 20002969
AU  - Keverne EB
TI  - The vomeronasal organ.
LA  - Eng
MH  - Action Potentials
MH  - Afferent Pathways
MH  - Animal
MH  - Behavior, Animal
MH  - Chemoreceptors/chemistry/*physiology
MH  - Female
MH  - GTP-Binding Proteins/metabolism
MH  - Human
MH  - Hypothalamus/physiology
MH  - Male
MH  - Neurons, Afferent/*physiology
MH  - Olfactory Bulb/physiology
MH  - Pheromones/physiology
MH  - Receptors, Cell Surface/chemistry/genetics/*physiology
MH  - Signal Transduction
MH  - Vomeronasal Organ/anatomy & histology/innervation/*physiology
RN  - EC 3.6.1.- (GTP-Binding Proteins)
RN  - 0 (Pheromones)
RN  - 0 (Receptors, Cell Surface)
PT  - JOURNAL ARTICLE
PT  - REVIEW
PT  - REVIEW, TUTORIAL
DA  - 19991105
DP  - 1999 Oct 22
IS  - 0036-8075
TA  - Science
PG  - 716-20
SB  - M
SB  - X
CY  - UNITED STATES
IP  - 5440
VI  - 286
JC  - UJ7
AA  - Author
EM  - 200001
AD  - Sub-Department of Animal Behaviour, University of Cambridge,
      Madingley, Cambridge CB3 8AA, UK. ebk10@cus.cam.ac.uk
RF  - 56
PMID- 0010531049
PID - 7933
SO  - Science 1999 Oct 22;286(5440):716-20
```

Figure 2: An Item from a MEDLINE Download.

entered from the keyboard. It treats the keyed-in text as arguments to LaTeX commands. You can, instead, add macro commands to an existing database or to one you download from the Net or get from a scanned image.

Add allusions to the manuscript.
As you write your manuscript, you tuck snips of information about particular references inside brackets. The program extracts these allusions, bounces them against the bibliographic database, extracts the matched records, orders them by accession number or Name-Year, and substitutes the order tags for the allusions in the manuscript.

Using TADS, this is the sequence of tasks that results in a database:

1. Write a file of instruction records. We will call it *Lbiblio.ins*, but you name it as you like. Each instructional record provides the directives to control the entry of a single field in what will eventually be a single database record.
2. Run *Lquegen*. It will use the information in *Lbiblio.ins*, together with your answers to its online questions, to write you a customized data entry program. We will call the source code for the data entry program *Lbiblio.c*. You can run *Lquegen* as often as you like, using different instructional files to develop different styles of database.
3. Compile *Lbiblio.c*. You don't have to be a programmer. A Make file is provided.
4. Run *Lbiblio*. It has sufficient flexibility to create databases for different scientific disciplines, each with its own control file, each of which has the format given it by *Lquegen*. Actually, if you know C, you can go into the source code and make some minor adjustments to change the format.
5. *Lbiblio* needs to know where to send the processed text. Suppose we call this file *Lbib.db*. The first time you run *Lbiblio* to create records for *Lbib.db*, it ships the start of a LaTeX file with a set of commands, one per field, to *Lbib.db*. The macro name for a field is based on the prompt for that field.
6. Each time you run *Lbiblio*, it will store the values for the options you choose in the control file, the TRL file, whose name you specify. In this example, it's called *Lbib.trl*. The TRL file written the previous run stores the correct starting accession number for the current run. And it keeps a log of each run.

Lbiblio.ins, a file of instructions

Prompts and macro names are specified by giving *Lquegen* an Instruction File *Lbiblio.ins* that describes the prompt features of the database entry program as a set of records. Each field that is part of the record structure in the eventual database requires a separate record in *Lbiblio.ins*. If there will be 10 fields in each database record, you need to write 10 records. Figure 3 has an example of an instruction file; it has 13 fields in each record.

The delimiters in *Lbiblio.ins* must be the same as the ones you will use for the final database. We use '~' as the record delimiter and '/' as field delimiter. Notice that the record is terminated by '~', not '/~'

Each Instruction File record must have 5 fields. You need not fill in all the fields. A field may be empty, *but it must end in a delimiter*.

1. FIELD NAME. This *must* be a single word. When the data entry program is run, this will be the main prompt. The field name will also be the field's LaTeX macro command name, so only alphabetic characters are acceptable. I've used upper case on the field names, but there's no particular reason to do so.
2. EXTRA. This is optional extra prompt text. It helps conformity, if different people are typing in the data. It can remind them about punctuation and/or word order. It is reproduced as written, tabs, spaces, whatever.
3. DEFAULT ANSWER: If you just press ENTER, this will be the default text. The default answer is copied to the database as written. You can override it by typing in text. If you want no text in the field in some record, type a space and then press ENTER.
4. SUBFIELD: Is this field to be subfielded? YES/NO.
5. AND: Is this subfielded field to be ANDed? YES/NO.

The last two fields require explanation. The database(s) that will be created will be structured so that each field is the argument of a specific LaTeX command, which operates on the whole field. To make small changes on the text, the less information you store in a field, the better. In *Lbiblio.ins*, notice that VOLUME and ISSUE, which usually are neighbors in a citation, are in separate fields. Unfortunately, we don't usually write individual authors and editors in separate fields; and most of the microvariation in print appearance between journals is precisely in these two fields. Using subfielding and `\AND` is an attempt to solve one problem: does the

```
AUTHOR/:[FirstField] (SUBFIELD.) ex:Brown, A.//YES/YES~
YEAR///NO/ NO~
TITLE/: Title of article, NOT of the book//NO/ NO~
PAGES/: Separate with hyphen. Ex: 164-169//NO/NO~
JOURNAL/: Name of Book-Journal. Default:'Tech Manual X234L'/
          Tech Manual 234L/NO/NO~
VOLUME/: '3' is the default/3/NO/NO~
ISSUE/: TM234L-0ZN-X34523 is the default/TM234L-0ZN-X34523/NO/NO~
ISBN///NO/NO~
EDITOR/: (SUBFIELD.) Name of editor. ex: A.B. Smith//YES/YES~
CITY/: city where book was published//NO/NO~
PUBLISHER///NO/NO~
BPAGES/: number of pages in book//NO/NO~
NOTES/ keywords, code words//YES/NO~
```

Figure 3: A File of Prompting Instructions

journal want an *and* before the last author or an '&' or nothing?

A subfielded field is divided into subfields, just as the record is divided into fields. Each subfield is terminated by a character you reserve as subfield delimiter. Notice that, in the example, the AUTHOR, EDITOR and NOTES fields are subfielded. When *Lbiblio*, the data entry program, prompts for text in a subfielded field, it repeats the prompt over and over again, until you press ENTER with no previous text. Subfielding has different uses — separating authors to facilitate indexing, separating titles and subtitles — but its usefulness here is that, with recycling, the program knows when you've typed the last author.

YES in Field 5 requests that the program insert an `\AND` before the last subfield in the field (`\AND` is a command we define; the user may redefine it later). It can only be used with subfielded fields. (In this example, it isn't used in NOTES.)

Lquegen interprets the records in *Lbiblio.ins* and writes out its understanding in a file called, in this case, *Lbiblio.ins.decode*. In a Linux-Unix environment, you can pause while running *Lquegen*, read *Lbiblio.ins.decode* and compare it to what you wrote in *Lbiblio.ins*. This is its analysis for fields 1 and 6 in our example design.

```
Record [1]:
    FieldID = AUTHOR
    extra = :[FirstField] (SUBFIELD.) ex:Brown, A.
    defaultans = (null)
    subfld = YES
    AND = YES

Record [6]:
    FieldID = VOLUME
    extra = : '3' is the default
    defaultans = 3
    subfld = NO
    AND = NO
```

Lquegen, a program that writes programs

Choosing delimiters Conflict between different programming systems that operate on a database is almost unavoidable. There is no set of symbols that are exclusively and universally reserved for program instructions, with non-intersecting subsets for the different programming languages. What is text in one language is a directive in another. It is a happiness-making happenstance when the sequencing of programs is such that while the text is being manipulated by one programming language, it is transparent to the others, until it is their turn. But it takes careful planning to avoid difficulties. Particularly in choosing delimiters.

First, you don't want to use a character commonly used in the text itself. This lets out the comma and maybe the colon and semicolon. Invisible characters are poor choices. Second, the program reserves control-X, control-Y, `{` and `}`. All other control characters are OK if LaTeX, your machine, compiler and/or script don't reserve them. Records from the database will be formatted by LaTeX, so `%`, `\`, `&` and `#` are very bad choices. Avoid `$`, `^` and backspace, which are used in LaTeX math mode.

Good delimiters for a database that will be LaTeX-processed are: `/`, `*`, `@`, `‘` (octal 140), | (which prints as a dash), `'` and `"` (single and double quotes). The characters `=` and `+` are OK if you don't bring in arithmetic values.

Adding ID fields It is useful to have a permanent identifier (ID) to tag each of the records in the

database that *Lbiblio* will prompt you to construct. *Lquegen* can add two fields at the end of the fields designed by the Instruction File (see the example in the Section on Database Record Format). They can be transferred to the top of the record by *genio:rearrange*, a TADS program that rearranges and outputs the fields you specify.

An accession number field.
This is an easy way to provide an ID for each record. The first record in the file is 1, the second 2 and so forth. Because you may eventually be merging several data files, it is a good idea to have the ID also indicate the source of the record or some other name that tells you instantly where the record came from. You can tell *Lquegen* what text should precede the accession number when it asks for Leading Characters; e.g., *Molbiol2000:* or *ClinStudyAA*. This is a permanent tag for the record. It is not the accession number that is eventually given to records used by any particular manuscript.

You may, if you wish, make all the accession numbers the same size. If you say you want a minimum width of 6, say, the program will pad each accession number with zeros to make it 6-digit wide. (You can write a lot of records before you overflow a 6-digit width.) If you use the default, the program won't pad the number. It will use the actual length.

A Name-Year ID field.
The program can construct an ID field using the name of the senior author and the year of the publication. You will need to tell it the fields where these items of information are to be found. Actually, *Lquegen* only knows that it is to use the text of the first field up to the comma by default; or up to whatever size you stipulate. It uses all the text of the second field. Case can be set for the name: set all letters uppercase, set all letters lowercase, or leave case as is.

The style of the ID will depend in part on the text you tell the program to insert between the two items; e.g., *Smith2000*, *Smith:2000*, *Smith,2000*, *Smith, 2000*, *Smith:-2000*. Or you can put a large piece of fixed text between the name and year. And, if you wish, you can select a width, so that the name is chopped or padded to conform.

You can also vary the appearance of the records when they are finally ensconced in the database file. You can start each field on a separate line, if you wish. And you can set line width. If the text you type in has no space (at least 1 complete word) within the requested line width, the program will add a hyphen to the end of the line prior to outputting it, and will alert you to the hyphen by causing the bell to ring.

Lbiblio, a data entry program

By the time you type in the last answer, *Lquegen* has written you the C source code for a data entry program called, in this example, *Lbiblio*. You compile it by running a *make* file that comes with the program:

```
make FILE=Lbiblio
```

Once compiled, *Lbiblio* is immediately ready to act as a prompter for text data that you enter through the keyboard and to do housekeeping chores such as adding LaTeX macro names and record/field delimiters.

Initializing the data entry program The program needs some specific information before it can start its work. These values can be declared as a set of options on the command line when *Lbiblio* is run. The minus sign is the signal that the next letter is an option. There is no space between the option and the value. Options are separated by spaces. There is no input file.

OPTION	VALUE	MEANING
-h	YES or NO	Want extra information?
-o	⟨*filename*⟩	The output database file
-q	integer	Starting accession number
-t	⟨*filename*⟩	The Control-Log (TRL) file

-h Is initialization help wanted?
If the answer is YES, the program provides short essays at the start of the run. The default is NO.

-o The name of the output file.
If the file doesn't exist, the program will create it, otherwise it will append to the file; it never overwrites existing text. So it is possible to come back time and again to the same file to add references. Or you can use the same program, if it is general enough, as most bibliographic records are, for databases from difference scientific disciplines, each in a separate database file. There is no default. If the program is creating an output file, it prepends a LaTeX skeleton file, which includes one macro in the preamble for each field in a record. The macro definitions can be modified to meet the font requirements of any particular journal or document structure. The skeleton LaTeX file is shown in Figure 4.

-q The next accession number.
It is assumed you will be adding to the database. To anticipate, it is a convenience that you can direct the program to get the value from the control file. But you can also write it on the command line.

-t The name of the TRL file.
Whenever the program is run, it writes the current values of the options to the TRL file, so that it can read them the next run. The program also records other data: how many records were created, the date and time, and characteristics of the data entry program: line width, accession number width, the file delimiters, which fields are subfielded, and so forth.

There are various ways to start the program. If you just write the name of the program on the command line, you will be in Interactive mode. This is the simplest way to jumpstart the program, but it takes the most time. If you use the command line, the options can be written in any order.

Interactively. Just type the name of the program; i.e.,

```
Lbiblio
```

The program will query you for the values of the options. The very first time you run the program, there is no Control File, so you need to name it interactively. If there is no file with the name of the database, the program will create it. And any time you want to start a brand new database with a new TRL file, run the program interactively.

From the command line. Type the options directly on the command line; for example,

```
Lbiblio -hn -oqmolbiol.db -q40 -tmolbiol.trl
```

means you don't want extra explanation, the database file is called *qmolbiol.db*, the TRL file is called *molbiol.trl* and the first record you write this run will have 40 as its accession number. If there is no file with the name of the database, the program will create it.

From the Control File. Type the name of the control file option on the command line as the only option; for example,

```
Lbiblio -tmolbiol.trl
```

will use the values stored in *molbiol.trl* the previous run. The accession number will be correct, because at the end of a session, the program writes the starting accession number for the next session to the TRL file. The TRL file also maintains a permanent record of the previous data entry sessions that involve the data entry program, the database and the TRL file.

You can call some of the values from the TRL file and override other TRL file values by giving the parameters new values on the command line; for example,

```
Lbiblio -tmolbiol.trl -hy
```

requests that all the previous options, those stored in *molbiol.trl*, be used, except this time, you'd like some help.

You can maintain several databases on different subjects, each with its own database file and its own TRL file.

What the data entry program does

- prompts for the necessary information for the field, using the prompt text from the Instruction File
- writes the predefined default answer for the field (if there is one) to the database, if you press ENTER. You can override the default answer by writing in other text. To get an empty field in a field that has a default answer, type a space and then press ENTER.
- adds the specified record and field delimiters to each record.
- ignores any record and field delimiters typed in the body of the record
- adds the subfield delimiter to subfielded fields. It adds the `\AND` command to subfielded fields, if that was requested.
- writes out the full record to the database file with the specified line width
- writes out the record as a paragraph or writes each field to a separate line
- appends a stylized and padded Accession Number field to the record, if this was requested in *Lquegen*. This permanent accession number should not be confused with the numberings that will be given to records in the file that contains the citation list for a particular manuscript.
- appends a Name-Year ID field to the record, using the name of the senior author and the year of publication, if this was requested in *Lquegen*.
- writes a LaTeX header to the top of a newly-created database. The header includes a command macro definition for each field in the record, where the macro name for that field is the user-specified prompt in the first field of *Lbiblio.ins*. And it prepends the same macro

```
\documentclass[10pt,letterpaper]{article}
\usepackage{alltt}
\usepackage{multicol}
\usepackage[dvips]{graphicx}
\usepackage{color}
\usepackage{boxedminipage}
\usepackage{pandora}

%PAGE/PARA LENGTHS
\flushbottom
\parindent=0pc
\setlength{\baselineskip}{14pt}
\setlength{\parskip}{13pt}
%PAGE STYLE
\setlength{\textheight}{7.4in}
\setlength{\textwidth}{5.5in}
\setlength{\oddsidemargin}{1in}
\setlength{\evensidemargin}{1in}
%HEADERS/FOOTERS
\pagenumbering{arabic}
\setcounter{page}{1}
\pagestyle{myheadings}
\markboth{}{Demo Bibliographic Database}

\newcommand{\etal}[1][et al.]{\textit{#1}}
\newcommand{\AND}[1][ and ]{\textup{#1}}
\newcommand{\AUTHOR}[1]{\textup{#1}}
\newcommand{\YEAR}[1]{\textup{#1}}
\newcommand{\TITLE}[1]{\textup{#1}}
\newcommand{\PAGES}[1]{\textup{#1}}
\newcommand{\JOURNAL}[1]{\textup{#1}}
\newcommand{\VOLUME}[1]{\textup{#1}}
\newcommand{\ISSUE}[1]{\textup{#1}}
\newcommand{\ISBN}[1]{\textup{#1}}
\newcommand{\EDITOR}[1]{\textup{#1}}
\newcommand{\CITY}[1]{\textup{#1}}
\newcommand{\PUBLISHER}[1]{\textup{#1}}
\newcommand{\BPAGES}[1]{\textup{#1}}
\newcommand{\NOTES}[1]{\textup{#1}}
\newcommand{\ACNUM}[1]{\textup{#1}}
\newcommand{\NAMEYEAR}[1]{\textup{#1}}

%ATTENTION: Before you process the file
%through Latex, make sure there is an
%\end{document} after the last reference.
%Remove any \end{document} in the body of
%the file.
\begin{document}
```

Figure 4: The Top of the Database File.

name to the start of that field in each citation in the database. See Figure 4.

What the data entry program does not do Aside from adding the `\AND` command, the program does not modify the text you key in. On the other hand, as you key in text, you can use your own macros to reduce typing time and errors: macro names for long journal names, an alias for an author with a long and difficult name. As an example, I've included a `\etal` command (see Figure 4), because it is usually italicized. And it is inconvenient to format it after the fact.

In general, unless you write for a single journal, it's almost impossible to write a canonical style for names. One strategy is to adopt a style that works fairly well for the journals in which you publish. LastName-Initial is more common than LastName-FullFirstName, so it's fairly safe to use that style. It doesn't, of course, prevent the need for small polishings: one journal wants last name, just initials; another wants last name, followed by initials with periods. I use periods, because they are easier to erase than to add. *Science* uses an Initials-LastName format, which makes alphabetizing on the field difficult. You can, however, alphabetize on the NameYear field. If you publish often in both in a *Science*-style journal and in one that uses LastName-Initial, it might be worthwhile keying in author fields in both versions. Depending on where you send the article, you will use one or the other field in the final List of References.

If you plan on making microadjustments to the AUTHOR field in Emacs or some other text processor, it is a good idea to customize your data entry program to write each field in the record to a new line. You then search on the command name.

What you can do in response to a prompt

Create the citation fields, one by one. In our example, once the program is initialized, it will ask a series of questions for each citation, where a citation can be a reprint, a book or a technical publication. The typed responses will be confined to separate fields. Answers may be of any length, including zero length; i.e., the field can be empty, but the program will add a field or record delimiter. Depending on the Instruction File, a field can be simple; i.e., the prompt is displayed and you type in text. Then the prompt for the next field is displayed. Alternatively, a hierarchical field prompt can be displayed, where the prompt is repeated again and again, each time defining another subfield. To stop a subfield and/or a field, don't type any text — just press ENTER.

Jump between fields in the record. You can jump between fields in the record by typing ^Y (control-Y). ^Y+6 or ^Y6 will jump forward 6 fields. ^Y-2 will jump back 2 fields. You can not jump out of a record. If you jump forward some large number,

you will land in the last field of the record. If you jump backward some large number, you will end up in the first field of the record. It is not advisable to jump from a subfielded field; it can mess up the record.

Jumping forward to the last field is useful when, as in the example database template, you've essentially completed the citation for an article and want to skip the BOOK questions. You can't jump past the record, because the program does its housekeeping in the final field of the record, including chopping a clumped record into lines of the width specified in *Lquegen*.

Jumping back repeats previous prompts. When the program jumps back, it does not erase the intermediate fields. It just starts prompting from whatever field it has jumped to. If used judiciously, this feature lets you recycle a cluster of fields. It is a way of creating ID-prepended fields, such as those in the MEDLINE download in Figure 2. However, the record is no longer well-structured as an ASCII-delimited file.

Stop the program. Type ^X to stop the program. It will stop immediately. The best place to stop the program is at the prompt to the first field, so that the previous record has been completely processed. If you stop in the middle of a record, you will lose some text, and the record will be incomplete.

Database Record Format Next is an example of two database records that were keyed in under the control of *Lbiblio*, which was itself created using the example instruction set. The first has a single author and no subfields; the second one has a subfielded AUTHOR field. Notice that the fields between the ISSUE and NOTES fields are blank. They don't apply to a journal article, so you would want to skip to NOTES. The number in the NOTES field was taken from the MEDLINE ID for the article. NOTES is also a good place to store the authors' first names, for the few times some publication will request full first names. And it can be utilized as a depository for keywords that can later be used for cross-indexing citations and sorting them by subject matter. The NOTES field can also serve to indicate the physical location of the reprint, editorial comments, and so forth. The two last fields were added by the program. Records are stored as shown. For publication, fields will be extracted, rearranged and beautified, using available TADS routines.

```
\AUTHOR{ Eisthen, H.L. }/ \YEAR{ 1992 }/
\TITLE{ Phylogeny of the Vomeronasal System
and of Receptor Cell Types in the Olfactory
and Vomeronasal Epithelia of Vertebrates }/
\PAGES{ 1-21 }/
\JOURNAL{ Microsc. Res. Tech. }/
\VOLUME{ 23 }/
\ISSUE{ 1 }/ / / / / /
\NOTES{ 93004928 }/
\ACNUM{DemoFile01}/
\NAMEYEAR{Eisthen, 1992} ~
\AUTHOR{ Freitag, J. @ Ludwig, G. @ Andreini,
I. @ Rossler, P. @ \AND Breer, H. }/
\YEAR{ 1998 }/ \TITLE{ Olfactory Receptors in
Aquatic and Terrestrial Vertebrates }/
\PAGES{ 635-650 }/
\JOURNAL{ J. Comp. Physiol. }/
\VOLUME{ 183 }/
\ISSUE{ 5 }/ / / / / /
\NOTES{ 99056834 }/ \ACNUM{DemoFile04}
/\NAMEYEAR{Freitag, 1998}~
```

Writing The Manuscript

This is a short manuscript that illustrates the technique for writing allusions, using ordinary wildcard syntax: a ? allows any single letter in the ? position; a * allows any amount of text or no text to intervene between the two neighboring text phrases. The spelling error in the last line is deliberate.

> Phylogeny of vertebrate pheromonic sensory systems is complicated by the proximity and similarity of the adjacent but distinct olfactory system [@??sthen*vertebrate*olfactory]. The presence of sex pheromone systems in goldfish and the anatomic analogies of distinct olfactory systems [@Dulka*sex*pheromone] clearly establish both the antiquity and the complex olfactory/brain relationships that seems to characterize most if not all vertebrates [@??sthen*microsc], [evolution@vertebrate@olfact]. The hypothesis that the Class II receptors are specialized for recognizing volatile odorants is questionable since some fish, e.g. Latimeria, possess both classes [Freitag*aquatic*vertebrate]. The presence or absence of an accessory olfactory bulb is not in and of itself sufficient to affirm or deny a functional vomeronasal system in a given species [@bhatnag?r*diversity*mammalian], [bhatnag?r*bats*phylogenetic]. Attempts to infer the form of the earliest vertebrate pheronomic structures by comparative anatomy of hagfish and lamprey are made difficult in that

the necessary physiologic data on these forms are not available [sennsory biology].~

All the text phrases must be found in the citation for a match, so an allusion is intrinsically a boolean AND. A difficulty with this technique is that the text phrases that make up any wildcard match word must be sequential. This is fine only if you remember the exact field sequence. So there is an alternate wildcard format: an @ prepends the word. The @ syntax tells the machine to match each phrase in the allusion from the beginning of the field.

I've elected to use square brackets, but any bracket pair will do to delimit the allusions. An option in *dork:keepbracketedtext* leaves the empty brackets in the manuscript, when it strips the text from the brackets. So the square-bracket format is good for Name-Year tags.

For a number tag, I'd still use the square bracket but I would write ([⟨*text*⟩]), and use the option that deletes the text brackets; the square bracket would be deleted, not the parens. ([@??sthen*vertebrate*olfactory]) might eventually be written as (6).

If an entire manuscript is to be searched for allusions, add a single, unique record delimiter at the end. The file can now be described to the program as a single-fielded single record. Size is not critical for journal articles. But one could encounter problems if you were to treat an entire book as a single record. However, I've dummied up a 7 megabyte record by repeating a real book several times. No problems were encountered.

Linking Database and Manuscript

Various TADS modules come into play. This is the general plan. Recall that the input file is *always* considered *read only*. Any modifications are reflected in the file where the processed records are shipped.

Using *dork:keepbracketedtext*, the allusions are stripped out and reappear in some output file, one per line,

The allusions become a list of wildcard match-words that *finder* compares to records in a single database or in a database that is a virtual merge of records from several databases.

What constitutes a good allusion? The answer is tautological: a good allusion is one that is sufficient to attract the reference you want and only the reference you want. In practice, these are some of the types of errors that are encountered when the Citation Allusion (CA) in the Manuscript (MSS) is pitted against the Citations List (CL) in the source database.

1. The citation is multiply listed in CL.
2. The citation is not listed in CL.
3. The citation is incorrectly written in CL.
4. The CA is incorrectly spelled in MSS.
5. The CA informational items are incorrectly concatenated.
6. The CA is inaccurate and retrieves no citation from CL.
7. The CA is so general it retrieves multiple citations.
8. There are multiple CAs to the same reference.

Depending on the circumstances, the citation is not retrieved or unrequested citations are retrieved or a correct citation is repeated.

You can increase the probability of accurate retrieval by utilizing LaTeX macro names. Instead of writing @199?*green*pheromone*goldfish, you would write @\date*199?*\author?green*\title{pheromone*goldfish. But some people object to doing this while in the throes of creative writing.

The next step — checking the allusions agains the retrieved citations — is crucial. And it requires your participation. *finder* ships copies of the citations it matches to a file. The key, i.e., the allusion that identified the bibliographic reference is prepended as a field to the citation.

If the number of citations extracted equals the number of CAs, it may mean we have no error. On the other hand, sources of error could balance out so that the number of CAs equals the number of citations. So we can not rely on counting the number of citations and simply comparing this to the number of CAs.

TADS can present the original list of allusions and the list of citations in ways that facilitate comparison of the lists. *genio:rearrange* will split off the allusions field to a separate file, so the original list of allusions and the allusions prepended to the retrieved citations can be compared. *alp* sorts lists alphabetically; using any of the fields as key. *squish* eliminates duplicates. Spelling errors may have to be corrected 'by hand'. When you are sure the corrected allusions and citations correspond in a one-to-one fashion, hand the text back to TADS. The program tags the final list of retrieved citations in an orderly sequence. You make two separate choices:

1. You chose between listing the references by first appearance in the manuscript or in alphabetical order. The same choice applies if you list references by chapter in a book.
2. In either case, you choose whether to number the records sequentially or by Name-Year ID.

The order tag — accession number or Name-Year ID — is substituted for the allusions in the manuscript. This is done in four steps.

1. The citations can be left in the order in which they appear in the manuscript. Or they can be alphabetized using *alp*. If they are alphabetized, it is usually by author field or by Name-Year ID.
2. If ordering is by Name-Year ID, the order was done in the previous step. You will, however, need to add a final letter to the ID, if there is more than one publication by the senior author for that year. Alternatively, *addtext:acnum* can add an accession number to each record. The accession number becomes the first field in the expanded record.
3. *genio:rearrange* creates a derivative database of records, each with two fields: the allusion and the accession number (or Name-Year). The allusion and number are now considered a substitution pair. The entire file of records has become a list of substitution pairs.
4. *addtext:sbstitute* brings in the file of substitution pairs and substitutes accession numbers or Name-Year IDs for allusions in the manuscript.

The List of Citations needs to be dressed up for publication. *genio:rearrange* extracts particular fields in each record in the order you specify. Case can be modified on a per field basis. Revamping the macro definitions in the preamble will take care of the general appearance of the font. A beautify program *addtext:mssformat* substitutes commas for (sub)field delimiters, periods for record delimiters. Currently, this step usually calls for some small adjustments in the text — such as adding or removing commas and periods in the AUTHOR field, adjustments that are not easy to make globally. I work in Emacs, using the macro name to get to the right field per record and make changes locally — changing case, deleting periods and transposing words, using Emacs commands. Writing the fields in the records on separate lines simplifies the work considerably.

The list of references is appended to the manuscript, `\end{document}` is inserted, and the manuscript is ready.

Conclusion

In dealing with a database you have designed yourself, there are non-trivial advantages to prepending a field name that is also a LaTeX macro:

1. A prompting program instills style conformity in the records of the database and reduces transcription errors.
2. You can design a canonical record style across multiple databases.
3. You can manipulate font size and shape by field by redefining the field macro in the preamble.
4. You can use field names as part of a wildcard search of the database to reduce errors in matching an allusion to the correct (read *wanted*) record in the database.
5. To effect small text changes while in a text editor such as Emacs, you can get to the right field by searching on the field name.

Bernice Sacks Lipkin