Standard File Headers

Nelson H. F. Beebe
Center for Scientific Computing
Department of Mathematics
University of Utah
Salt Lake City, UT 84112
USA
Tel: +1 801 581 5254
FAX: +1 801 581 4148
E-mail: beebe@math.utah.edu

06 March 1996
Version 1.28

Contents

%

List of Tables

Licensing information

The program currently being distributed that relates to standard file headers is contained in the file '`filehdr.el`'. It consists of numerous support functions for to the creation and maintenance of file headers. This program is *free*; this means that everyone is free to use it and free to redistribute it on a free basis.

Specifically, we want to make sure that you have the right to give away copies of the programs that relate to '`filehdr.el`', that you receive source code or else can get it if you want it, that you can change these programs or use pieces of them in new free programs, and that you know you can do these things.

To make sure that everyone has such rights, we have to forbid you to deprive anyone else of these rights. For example, if you distribute copies of the file '`filehdr.el`', you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must tell them their rights.

Also, for our own protection, we must make certain that everyone finds out that there is no warranty for the programs that relate to '`filehdr.el`'. If these programs are modified by someone else and passed on, we want their recipients to know that what they have is not what we distributed, so that any problems introduced by others will not reflect on our reputation.

The precise conditions of the licenses for the programs currently being distributed that relate to '`filehdr.el`' are found in the General Public Licenses that accompany them. The programs that are part of GNU Emacs are covered by the GNU Emacs copying terms (see section License in *The GNU Emacs Manual*), and other programs are covered by licenses that are contained in their source files.

Chapter 1

Background

With the rapid spread of the global Internet, which by 1991 reaches more than a half-million computers all around the world [3], the opportunities for free exchange of software and textual data are greatly enhanced.

While this brings exciting new capabilities to many people, not just those involved in academic research, it is hampered by several factors.

First, not all network file exchange is error-free. Electronic mail systems in particular are notorious for corrupting information, either by truncation of lines or message bodies, or by transliteration or other altering of certain characters. These problems are most severe for mail exchanges *between* major networks, such as between the Internet and Usenet or Bitnet.

Second, no standards yet exist for describing the contents of files. While this is an area of research at some academic institutions, the wide variety of operating systems in use, and the growing numbers of computers (approaching 100 million on a world-wide basis in 1991), suggest that such standards may never exist, any more than products on the commercial market, from soup to saltines, have standard labels.

Third, without a record of origin of software and data, it is impossible for users to verify that they have up-to-date copies, or to contribute improvements and additions back to the original authors.

Fourth, without a standard means of encoding information in file headers, there is no hope of automating the process of collecting information from file headers to produce enhanced file archive summaries, catalogs, and the like.

During the author's 1991–92 tenure as President of the TEX Users Group, efforts were undertaken to improve the quality and quantity of electronic distribution of TEX-related software and data. While this work had a narrow focus, it has quite general ramifications, and the GNU Emacs

support code here is quite general, and capable of handling almost any type of computer-readable textual material.

It does *not*, however, address the issue of exchange of binary (non-textual) data; that has a number of difficulties associated with it, the two most severe being rigid formats intolerant of extension, and machine-specific encoding and byte order.

During a visit to Heidelberg University in June 1990, the author spent a pleasant brain-storming session that lasted until 3am with a dozen colleagues (who names, alas, were unrecorded) from Heidelberg, Mainz, Darmstadt, and Goettingen.

We discussed many things that evening, but one topic in particular led to this work: an informal proposal for standard file headers that could address all of the problems noted above.

Chapter 2

What's in a header?

The BibTEX system for support of bibliographic data bases was developed by Oren Patashnik at Stanford University, based on earlier work by Brian Reid at Carnegie Mellon University on the Scribe document formatting system [4]. BibTEX is described in Leslie Lamport's book [2] on LATEX. It is based on the notion that bibliographic items can be divided into distinct *classes*: articles, books, reports, theses, and so on.

Each class of documents has certain features in common. For example, journal articles have authors, titles, volume numbers, often issue numbers, page numbers, and dates of publications. Theses and reports would have the name of an institution attached.

The number of classes of documents is not fixed; indeed, it may change with time, or between cultures and languages. Thus, a bibliographic system must be *extensible*. BibTEX provides this critical feature by an implementation in a programming language that knows how to parse the general structure of a bibliographic data base entry, without particular knowledge of the classes, or attributes of classes. That information is instead encoded in a *style file*, which is written in a much more compact form that is specialized for its job, and is presumably easier for users to change than BibTEX itself is.

The style file can specify which attributes are required to be present in a class (e.g. a Ph.D. thesis must have an institution), and which attributes are optional (a book may or may not have an International Standard Book Number, ISBN).

Some styles may not require all attributes in a particular class, so BibTEX simply *ignores* attributes not required by the current style, checking them only cursorily for proper syntax.

In addition, the style file can specify how individual bibliographic en-

tries extracted by BibTEX from data base files are to be formatted. In a typesetting application, this flexibility is important, because there are a great many bibliography formatting styles, and each journal or publisher often has rather strict (and arbitrary) rules that authors must adhere to.

How does this relate to the question of file headers?

Clearly, the notion of classes and attributes applies to all computer files as well. The class is the file type, such as Lisp file, Pascal code file, and national census data file. The attributes are things like author(s), author's address, date of last modification, file name, revision history, character set name, and so on.

In many operating systems, file naming conventions have been adopted by which the name encodes information about the class to which the file belongs. For example, if the file name ends in '`.c`', it is assumed to contain code written in the C programming language. Unfortunately, few file systems are general enough to permit the creators of computer files to encode additional header information that might be more detailed.

Since this additional information cannot be standardly encoded in the file system, it must be supplied in some way inside the files themselves. This is not universally possible, particularly with binary files.

However, textual data tends to be much more portable between computer systems, and all reasonable programming languages and text processing systems make some provision for *comments*, that is, explanatory material inserted into the file which is otherwise ignored by the program which processes the file.

Such comments are generally identified by a unique start symbol, followed by the comment text, and a unique end symbol.

The start symbol is usually a particular special character, or special short character sequence, not otherwise required in the language in which the file is encoded. Sometimes the start symbol must begin in a certain column of the line, such as Fortran's `C` or `*` in column 1, or is implicitly present at a certain column (assembly languages for older computers often decreed something like "a comment starts in column 32 of the input record").

The end symbol is frequently an end-of-line condition, which need not be an actual character. This convention is simple, but limits comments to single lines. If a comment end symbol other than end-of-line is chosen, the comment body may span multiple lines. Thus, the PL/1 and C programming languages delimit comments by `/*` and `*/`, and Pascal by `(*` and `*)`, or by paired braces. Some programming languages even permit comments to be properly *nested*, so that one can comment out a block of code that itself contains comments.

Ideally, a comment syntax should be simple, yet permit *any* processor-representable characters to appear in the comment text, so as not to hinder

freedom of expression.

In any event, with most programming languages, we should be able to encode file header information as comments in such a way that expression is not restricted, yet both humans and suitable computer programs can recognize the presence of the file header.

Chapter 3

Putting it all together

The preceding sections have outlined the notions of *classes*, *attributes*, and *comment embedding*. What we want to do is to borrow the syntax used by BibTEX for bibliography data base files, and encode the file header as comment body text in whatever syntax the programming language allows, but to do so in such a way that it can be readily recognized by both humans and computer programs.

Thus, in a Fortran file, for which comments run from a `C` in column 1 to end of line, our file header might look something like this:

```
C     @Class{
C       attribute1 = "value1",
C       attribute2 = {value2},
C       attribute3 = {value3 with {extra braces}},
C       attribute4 = {value4 with "quotation marks"},
C       attribute5 = "value5 with ""quotation marks""",
C       ...
C     }
```

The key to programmatic recognition of the header is the syntax *name followed by an opening brace, zero or more attribute assignments, and a closing brace.* The attribute value fields can be enclosed in quotation marks, or balanced braces, as shown above.

In the event that braces otherwise have special significance (such as in one of Pascal's comment forms), other distinct paired delimiters could be used; in the ASCII character set, this means parentheses, square brackets, or angle brackets.

The order of attributes is significant only in the event of duplications; in such a case, the *last* value assignment is the one to be used. Conventions

for the order of attributes will make file headers easier to read, however.

Readers familiar with BibTEX will note the absence of a *tag* following the opening brace. In the bibliography data base application, the tag serves as a unique citation key that can be placed in other documents to uniquely identify the bibliographic reference. In the current application for file headers, we have no need of such a tag.

For languages in which comments continue from a start symbol to end of line, it will be useful, though not essential, to make the comment section containing the file header more visible. This can be done in a variety of ways, such as by doubling or tripling the comment start symbol, or putting a distinctive character sequence, like several asterisks or an arrow, `==>`, after it. The essential point is that if each line begins with a comment start symbol, *that same prefix must be used on every line of the header.* Not only does this enhance visibility, it makes it possible for a relatively simple computer program to identify the first line of the header and recognize the comment syntax automatically, and then collect the remainder of the header by discarding identical comment prefixes from succeeding lines until a complete header has been collected.

Chapter 4

Outline of file headers

This chapter briefly describes what the file headers contain: class names, attribute names, and attribute values. Each is treated in a separate section. Detailed descriptions of attributes will be found in the next chapter (see 5 [Attribute descriptions], page 15).

4.1 Class names

What should the class name in a file header be? We want it to be indicative of the file contents, even to a reader unfamiliar with the computer system from which it originated. Here are some desirable criteria:

- The class name should *not* be restricted by the length constraints of many file systems, and it should not use abbreviations, because they are often unintelligible to readers unfamiliar with the originating computer system, or with the language in which the header is written.

- It must also be possible to generate the class name automatically from knowledge of what the file name is, at least for those many classes of files that are distinguished by particular phrases in their file names.

- Class names must be standard across different operating systems, so that when files are moved between such systems, they can be readily associated with the correct class.

- Class names must be recognizable by a simple computer program, and thus must conform to an agreed-upon syntax.

I therefore propose that class names consist of an optional at-sign, `@`, immediately followed by an initial letter, optionally followed by letters, digits, and hyphens, followed by the phrase `-file`.

Letter case may be mixed for readability, *but is not otherwise significant*: `@LATEX-FILE` and `@LaTeX-file` represent the same file class.

This style of naming is common to many programming languages. Hyphens between words improve readability, while avoiding ambiguities introduced when spaces are allowed to be part of names.

4.2 Attribute names

What file header attributes do we need? Here are several that are desirable: abstract, author(s), checksum, code table, date, documentation, filename, keywords (for later indexing and cross-referencing), postal, electronic mail, and WorldWide Web addresses, and version.

Attribute names have the same syntax as class names, except that an at-sign, `@`, is never present. New attribute names can be added as needed, with the understanding that the file header processing software will ignore attributes that it has not been programmed to deal with.

4.3 Attribute values

What about attribute values? These are for the most part arbitrary text strings, usually delimited by quotation marks. In the event that quotation marks are needed in the text itself, braces (or parentheses, square brackets, or angle brackets) may be used instead, provided that they are properly nested. The value text should *not* presuppose the existence of any particular text formatting system; in particular, it should be understandable to a human reader when it is displayed in the 95 printable characters of the ASCII character set.

Attribute values may span multiple lines, and in most cases, newlines can be treated like spaces. However, file header processing software *must* distinguish between spaces and newlines, and in some cases, such as for address values, newlines will be preserved in the output.

Since file headers are encoded inside language comments, each line will often begin with a comment start symbol and white space chosen to provide neat formatting of the header to enhance its readability. Thus, after stripping the comment start symbol, leading white space (blanks and horizontal tabs) may be ignored.

File header processing software *may* choose to eliminate common prefix strings consisting of a comment start symbol and following white space from successive lines of a single value, but preserve additional indentation space. Thus, the input

```
;;; name = "Blah blah blah blah blah blah
;;;         blah blah blah blah blah blah.
;;;
;;;              Blah blah blah blah blah.
;;;         blah blah blah blah blah blah.
;;;
;;;              Blah blah blah blah blah."
```

could produce the value string

```
Blah blah blah blah blah blah
blah blah blah blah blah blah.

     Blah blah blah blah blah.
blah blah blah blah blah blah.

     Blah blah blah blah blah.
```

if common prefixes are stripped, or

```
Blah blah blah blah blah blah
blah blah blah blah blah blah.

Blah blah blah blah blah.
blah blah blah blah blah blah.

Blah blah blah blah blah.
```

if all leading white space is discarded.

BibTEX adopts that convention that braced groups inside a value string are protected from certain actions, such as letter case conversion, or sorting. In particular, a single quotation mark may be enclosed in braces to prevent its recognition as a value string terminator, assuming the string was started by a quotation mark. Since BibTEX expects that its output will be processed by the TEX typesetting system, where braces serve as grouping commands, and are not normally themselves printable, this is a reasonable choice: the value string `"A quotation mark, {"}, must be braced"` will be reduced by TEX to `A quotation mark, ", must be braced`.

In the context of general file headers, this convention is not reasonable, because the value strings will not in general be processed by TEX, but instead, will be treated as verbatim strings.

Similarly, although the C programming language has character escape conventions to permit encoding of non-printable characters in printable form, such as `\n` for newline and `\t` for horizontal tab, such usages are undesirable in the context of general file headers that must serve for many different programming languages and file types.

Several programming languages adopt the convention that a quote inside a quoted string is represented by an adjacent pair of quotes. This convention is easy to understand, requires no additional escape characters, and permits unrestricted representation of all printable characters, and of course, white space (blanks and horizontal tabs). We adopt this convention for attribute value strings, but note that since balanced braces (parentheses, square brackets, angle brackets) can also be used to delimit value strings, the need for such doubling will be rare.

Chapter 5

Attribute descriptions

In this chapter, we go into the details of each of the currently-defined attributes in a standard file header. Attributes are treated in alphabetical order in the following sections; they need not occur in that order in file headers.

5.1 Abstract

The `abstract` attribute can supply a short abstract string to complement the longer `docstring` entry. This should normally be limited to a single paragraph.

For example, large research institutes often prepare an annual publication list with abstracts of documents prepared by staff members. With care in the preparation of the file headers, and suitable software support, much of that annual report could be extracted automatically from the file headers.

5.2 Address

The `address` attribute should have a postal address. Be sure to include a country in your address; your file may be shared with users all around the world.

Here is an example from the file header for this document:

```
%%%     address         = "Center for Scientific Computing
%%%                       Department of Mathematics
%%%                       University of Utah
```

```
%%                      Salt Lake City, UT 84112
%%                      USA",
```

5.3 Author

The `author` attribute should give the full name of the author, in the order as it is conventionally spoken. In much of the Western world, the family name goes last.

If there are multiple authors, separate them by the word `and`, rather than by commas. The reason for this is that BibTEX has special algorithms that use this convention to allow parsing of names in some foreign languages, as well as names with qualifiers, like `Jr.`, and those algorithms could be adapted by other programs that process file headers. Even simple programs could separate the names by splitting at the word `and`.

Here is the `author` attribute from this document's file header:

```
%%      author          = "Nelson H. F. Beebe",
```

5.4 Checksum

The background chapter (see 1 [Background], page 3) noted that it is important to be able to verify the correctness of files that are moved between different computing systems. The way that this is traditionally handled is to compute a number which depends in some clever way on all of the characters in the file, and which will change, with high probability, if any character in the file is changed. Such a number is called a *checksum*.

Good algorithms for computing checksums are not obvious. One possibility is to count up the number of characters, words, and lines; in the UNIX world, this is easily done with the `wc` program. Another possibility is to just add up the numerical values of all the characters and use the resulting sum as the checksum. Both of these would change if characters were added or removed, but they would not change under transposition of characters, words, or lines.

Consequently, a lot of research has been done on algorithms for finding checksums, and some have even achieved international standardization. One of these standard algorithms is known as a CRC-16 checksum. CRC stands for *cyclic redundancy checksum*, and the redundancy of following it with the word *checksum* is accepted practice. The CRC-16 checksum is capable of detecting error bursts up to 16 bits, and 99 percent of bursts greater than 16 bits in length. The checksum number is represented as a 16-bit unsigned number, encompassing the range 0 ... 65535. Thus, there

is roughly one chance in 65535 of an error not being detected, that is, of two different files having the same checksum.

Of course, no human should have to compute a checksum; that is a job for a computer program. The GNU Emacs support software described in this document handles the job for you.

We cannot use just any checksum program, however, for several reasons:

- The checksum program must itself be portable and freely available, because verification of the checksum may be required on any machine that the file is transported to.

- File formats change from system to system. On some file systems, text files are represented by fixed-length records. On others, variable length records include a count of the number of characters in each line. On still others, lines end with character terminator sequences like CR, LF, or CR LF.

- The file must contain the checksum, but somehow, the checksum itself must not be counted when the checksum is computed. Otherwise, we could never achieve self-consistency: each insertion of a new checksum would change the checksum.

- Because of the varying line representations in file systems, trailing blanks should not be included in the checksum. Such blanks waste space, and should never be significant; they can be lost when text is refilled in a line-wrapping editor, or during electronic mail transmission. It is a good idea to get rid of them; the Emacs file header maintenance functions described elsewhere (see 6 [GNU Emacs editing support], page 27) do this for you automatically.

- Horizontal tabs look like spaces on the computer display, but are really separate characters. They are often subject to translation to spaces by electronic mail systems. For most text files, you can safely replace them by blanks, which is easy to do in Emacs: just mark the whole buffer with `C-x h`, and then type `M-x untabify`.

  UNIX `Makefile`s and `troff` files are notable exceptions to this; tabs are *significant* and cannot be replaced without destroying the meaning of those files. That is why the GNU Emacs file header maintenance functions never touch tabs.

These considerations make it clear than existing software for computing checksums just will not do. I raised these points in an editorial challenge

[1] in the TEX Users Group journal, TUGboat, and in the spring of 1991 received a clever solution from Robert Solovay at the University of California, Berkeley.

Solovay's program, called simply `checksum`, is written in a literate programming language called CWEB. The output is C code that conforms to the 1989 ANSI/ISO C Standard. In computing the checksum, it ignores line terminators, and any previous checksum, and since it has been placed in the public domain, it solves all of the problems noted above. Besides a CRC-16 checksum, it also produces counts of characters, words, and lines. In the event that `checksum` has not yet been installed, this information can be compared against the output of the UNIX `wc` utility. `wc` is simple enough that it can easily be reimplemented on any system.

`checksum` also has an option to verify the correctness of the checksum in a file; you could use this to check for corruption after transferring a file with standard file headers to your system.

Although `checksum` can be run manually, the GNU Emacs support code does it for you, producing an entry in the file header that looks something like this:

```
%%%     checksum        = "25868 849 3980 28305",
```

The four numbers are the CRC-16 checksum, line count, word count, and character count. You must remember that the character count will change if the file is stored with different line terminator conventions; the other numbers will remain constant.

5.5 Codetable

In the computing world of the 1990s, two major character sets are in wide use: EBCDIC on IBM mainframes and their clones, and ISO/ASCII on everything else. EBCDIC is an 8-bit character set, offering characters in the range 0 ... 255, while ISO/ASCII is a 7-bit character set, with characters in the range 0 ... 127. On most machines, ISO/ASCII text is stored in 8-bit characters.

In turns of numbers of computers, ISO/ASCII is by far the most common, since it is the character set used by all personal computers and workstations.

Unfortunately, a 128-character set with 95 printable characters and 33 control characters is inadequate for most non-English languages. Many European languages require accented characters or additional letters, and Chinese, Japanese, and Korean have thousands of pictographic characters.

Consequently, computer vendors have dealt with this by offering ISO 'code pages' — variations in the encoding of characters 128 ... 255, and sometimes even in the encoding of punctuation characters in the range 0 ... 127.

Standards bodies are actively working on the development of a new character set that will support all, or almost all, of the world's present and past languages. One of these efforts is a 16-bit character set called Unicode, and another is a 32-bit character set called ISO 10646. Efforts are now underway to merge these efforts into a character set called ISO 10646M (M for merged).

Given the speed at which committees work, and the enormous impact on millions of computers, and people, of a change in text encoding, it seems unlikely that the impact of these efforts will be felt for another decade.

The code page problem, however, does have to be dealt with. The standard file headers provide for this with an attribute entry like

```
%%%     codetable       = "ISO/ASCII",
```

If the file is encoded in, say code page ISO-8859-3, then the header could say that:

```
%%%     codetable       = "ISO-8859-3",
```

Of course, if an ASCII file were transferred to a system with EBCDIC, the file would not be immediately readable until the character values were translated to EBCDIC. The checksum described in the preceding section would be incorrect, but at least the fact that the file header stated that the code was originally ISO/ASCII would explain any translation peculiarities that cropped up later.

The attribute name `codetable` was chosen over `codepage` because the latter notion is restricted to variants of ISO/ASCII.

5.6 Date

Computer files should always carry a date-and-time stamp to record time of the last modification. Some file systems even store date-and-time stamps for last read, last write, last backup, and so on.

Unfortunately, many computers do not have a reliable time standard, and if they lack a network connection, have no way of maintaining a correct one. Date-and-time stamps are recorded in the file system, rather than the file itself, and are usually lost when the file is transferred to another system. That is regrettable, but it is a fact of life we still have to tolerate.

Consequently, a standard file header should carry a date and time. The editing support described here supplies it in the form

```
%%%     date            = "07 Oct 1991",
```

Dates and times are expressed in a variety of formats that depend on the country and culture. Some software can deal with a considerable variety of formats, ranging from "last Wednesday" to "1991.11.06:12.34.17". The important point is that the encoding *must be unambiguous*. In particular, forms like `12/06/91` should be avoided: does it mean the 12th day of the 6th month, or the 6th day of the 12th month? The year should *not* be abbreviated to two digits; the new millenium is not far away.

5.7 Docstring

For the purposes of cataloging files, and recognizing their contents, it is helpful to have a few paragraphs of description. This is provided for by the `docstring` attribute, which might look like this:

```
%%%     docstring       = "This LaTeXinfo document describes
%%%                       filehdr.el, a GNU Emacs support
%%%                       package for the creation and
%%%                       maintenance of standard file
%%%                       headers, such as this one.  It
%%%                       may be processed by LaTeX to
%%%                       produce a typeset document, or by
%%%                       M-x latexinfo-format-buffer in
%%%                       GNU Emacs to produce an info file
%%%                       for on-line documentation.
%%%
%%%                       The checksum field above contains
%%%                       a CRC-16 checksum as the first
%%%                       value, followed by the equivalent
%%%                       of the standard UNIX wc (word
%%%                       count) utility output of lines,
%%%                       words, and characters.  This is
%%%                       produced by Robert Solovay's
%%%                       checksum utility.",
```

This documentation need not be a user's manual for the file, unless the necessary information can be communicated in a few paragraphs of no more than a couple of thousand characters. Think of it instead as an extended abstract.

Someday, we may have tools that will extract documentation strings from standard file headers and turn them into catalogs.

5.8 Email

People who exchange computer files now often have network access, and the worldwide Internet is growing rapidly. It will not be long before network connections are as commonplace, and important, as telephone connections now are. Most networks support electronic mail, and the trend is to develop uniform addressing schemes that will work the world over. Thus, an electronic mail address, when available, is as important as a postal address for the author(s).

Here is an example:

```
%%%     email           = "beebe@math.utah.edu (Internet)",
```

Since there are several networks in existence, with different naming conventions, it is helpful to identify the network as in this example.

In the event that there are multiple authors, electronic mail addresses should be given in the same order, separated by the word `and`, just the way the author attribute value is coded. Of course, not all of the authors might have such an address, so additional qualification, such as by a parenthesized set of initials, could follow each address. Use your ingenuity, but in such a way that someone you've never met will still understand what you mean.

5.9 FAX

The `FAX` attribute should be formatted just like the `telephone` entry. Here is an example:

```
%%%     FAX             = "+1 801 581 4148",
```

FAX machines are now very commonly used in business throughout the world, so if you have such a facility, it is a good idea to include it in the file header.

5.10 Filename

Different computing systems have different file naming conventions; in particular, there are significant variations in the naming of files. Some systems, like the Apple Macintosh, permit arbitrary strings of characters, including

blanks. Others, like MS DOS on the IBM PC and clones, limit names to two parts, a base name and an extension, or type, with the two separated by a period (dot, full stop).

File headers should therefore carry an indication of the original name of the file, and if the file is expected to be referenced by other files, then it is *imperative* that the name chosen be representable on a wide variety of, and preferably all, computing systems. Today, this in practice means the 8-character base name and 3-character file extension of MS DOS, which runs in tens of millions of personal computers. There are still a few survivors of older operating systems with more stringent requirements on file names, but they are obsolete and rapidly disappearing.

The filename should be case *insensitive*, and in the header, spelled in lower-case letters. It should start with a letter, and use only letters, digits, and perhaps, hyphens (minus signs) in the rest of the name, with no more than a single period in the name.

This document's file header contains the attribute entry

```
%%%     filename        = "filehdr.ltx",
```

`filehdr` is an abbreviation for "file header", and `ltx` for "LaTeX", the name of the document formatting system that typesets this document.

5.11 Keywords

Large archives always pose a search problem for human users, and it has long been traditional to try to classify members of the archives by *keywords* that might come to mind when someone is searching for the file. Some journals have standard sets of keywords to classify articles by, and include them near the abstract of each paper.

With standard file headers, the range of possible keywords is enormous, and authors will just have to be diligent about finding good sets of descriptive keywords. They should appear in the attribute value as phrases separated by commas, as for this document:

```
%%%     keywords        = "file header, checksum",
```

5.12 Supported

All computer files reach a stage of stagnation, where for various reasons, their authors no longer maintain them. Nevertheless, it is helpful to know whether the author of a given file is interested in hearing of problems or comments, and the file header can say so by an entry like this one:

```
%%%     supported       = "yes",
```

If it says `yes`, this does not provide any guarantee that any problems reported will be fixed, but just that the author's intentions are good, and reasonable efforts will be made to do so. Some authors even care so much about their work that they offer monetary rewards for reports of bugs and errors.

If it says `no`, then you are on your own, because the author never wants to hear from you on the subject of this particular file.

Other attribute values can be readily imagined, like `only for money, cash in advance`, but a simple `yes` or `no` is probably adequate for most people.

5.13 Telephone

The `telephone` attribute should include the area code with telephone number. If there are multiple values, separate them by commas. Here is an example from the file header of this document:

```
%%%     telephone       = "+1 801 581 5254",
```

Use the international form of the number, including the country and city/area code.

5.14 Time

The `time` attribute should be of the form `hh:mm:ss`, or if a time zone abbreviation (say, `GMT`) can be found, `hh:mm:ss GMT`. It is recorded separately from the `date` to ease the parsing job of software that processes file headers.

Here is a typical example:

```
%%%     time            = "18:02:38 MST",
```

5.15 URL

Since its introduction in the early 1990s, the WorldWide Web has spread rapidly, so that most public interest in the Internet is associated with it, and so that most Internet sites that previously had electronic mail, ftp, and telnet services, now also have a WorldWide Web presence.

The Uniform Resource Locator, or URL, is therefore a suitable addition to the standard file headers; the one in this file looks like this:

```
%%%     URL             = "http://www.math.utah.edu/~beebe",
```

Since most sites have found it convenient to name a particular machine with the prefix "www.", from an electronic mail address one can often guess what the corresponding URL should be. Nevertheless, the host with that name is often different from the login host, so the Emacs code in '`filehdr.el`' may not successfully identify it automatically. Thus, you can provide an overriding private definition like this in your '`.emacs`' startup file:

```
(setq file-header-user-URL "http://www.math.utah.edu/~beebe")
```

5.16 Version

Computer files created by humans almost inevitably go through many revisions, whether they are programs to control a satellite, or just the words of a promotion for the latest soap product.

Computer vendors have long dealt with this by attaching *version numbers* to software releases. These consist of two or three numbers with some separator character, such as a period (full stop, dot). The first number is called the *major version number*; it gets changed only at long intervals, usually years, when really significant changes have been incorporated. A second number is a *minor version number* which is incremented as smaller changes and bug fixes are incorporated. Sometimes a third number is appended, which is an *edit number*; it gets incremented every time any change at all is made to the file.

In careful software production, a change log is kept to record the reasons for every change; this is particularly important when commercial interests or legal issues are at stake. [Military organizations the world over are famous for their paperwork trails; perhaps that is what helps to keep them busy during times of peace.]

For smaller files, you can probably get by with just a major version number and an edit number; for larger projects, three or more are recommended.

Here is what one version of this document had in its standard file header:

```
%%%     version         = "1.01",
```

Version numbers are particularly useful when reporting problems to the author of a file; they allow rapid verification of whether the author and end user are even talking about the same thing.

5.17 Multiple values

Keywords like `author` and `address` may be inadequate for files prepared by more than one person. If several authors share a common address, then using the keyword `and`, to separate names in the `author` field is unambiguous. However, if the postal address, electronic mail address, telephone number, and FAX number vary, it is advisable to clarify the header by attaching a hyphen and a numeric suffix to the attribute name. Here is an example:

```
%%%     author-1        = "Marie Claire LeBrun",
%%%     author-2        = "Hans Peter Brun",
%%%     author-3        = "Jill Brown",
%%%     address-1       = "...",
%%%     address-2       = "...",
%%%     address-3       = "...",
%%%     email-1         = "...",
%%%     email-2         = "...",
%%%     email-3         = "...",
%%%     telephone-1     = "...",
%%%     telephone-2     = "...",
%%%     telephone-3     = "...",
%%%     FAX             = "...",
```

File-header parsing software must be prepared to handle numeric suffixes like this for any keyword. If a keyword doesn't have such a suffix, as the `FAX` keyword in this example, then it should be assumed to apply to all authors.

Chapter 6

GNU Emacs editing support

The preceding chapters have outlined the background for, and contents of, standard file headers. Here we show how to generate them with very little effort.

The GNU Emacs file '`filehdr.el`' contains the following user-callable functions:

```
make-file-header
show-file-header-variables
test-file-header
update-checksum
update-date
update-date-and-minor-version
update-file-header-and-save
update-major-version
update-minor-version
update-simple-checksum
```

There are several other functions in that file, but they are for internal use only, and will not be further documented here.

When you want to add a new file header to an existing file, you just type `M-x make-file-header`; this produces something like this at the top of your file:

```
%%% ====================================================================
%%%  @LaTeX-file{
%%%     author          = "Nelson H. F. Beebe",
```

```
%%%     version         = "1.28",
%%%     date            = "06 March 1996",
%%%     time            = "13:14:03 MST",
%%%     filename        = "filehdr.ltx",
%%%     address         = "Center for Scientific Computing
%%%                        Department of Mathematics
%%%                        University of Utah
%%%                        Salt Lake City, UT 84112
%%%                        USA",
%%%     telephone       = "+1 801 581 5254",
%%%     FAX             = "+1 801 581 4148",
%%%     URL             = "http://www.math.utah.edu/~beebe",
%%%     checksum        = "53883 2543 10843 81774",
%%%     email           = "beebe@math.utah.edu (Internet)",
%%%     codetable       = "ISO/ASCII",
%%%     keywords        = "file header, checksum",
%%%     supported       = "yes",
%%%     docstring       = "This LaTeXinfo document describes
%%%                        filehdr.el, a GNU Emacs support pack-
age for
%%%                        the creation and maintenance of standard
%%%                        file headers, such as this one.  It may be
%%%                        processed by LaTeX to produce a typeset
%%%                        document, or by M-x latexinfo-format-
buffer
%%%                        in GNU Emacs to produce an info file for
%%%                        on-line documentation.
%%%
%%%                        The checksum field above contains a CRC-
16
%%%                        checksum as the first value, followed by the
%%%                        equivalent of the standard UNIX wc (word
%%%                        count) utility output of lines, words, and
%%%                        characters.  This is produced by Robert
%%%                        Solovay's checksum utility.",
%%%  }
%%% ====================================================================
```

Where does it get all of this information? Well, the file name, date and time stamps, author name, electronic mail address, and date are all determined automatically from calls to various system services. For example, on UNIX, the author name comes from the file '`/etc/passwd`'; on VAX VMS, it will

come from the file 'SYS$MANAGER:SYSUAF.DAT'.

The comment syntax was determined from the file extension, and we'll say more about it later.

The only information above that Emacs cannot determine is your postal address, and telephone and FAX numbers, and possibly, your WorldWide Web URL. These only have to be supplied once, usually in your GNU Emacs startup file, '.emacs'. This is most easily done with Lisp code that looks something like this:

```
(setq file-header-user-address  ; for M-x make-file-header
  "Center for Scientific Computing
Department of Mathematics
University of Utah
Salt Lake City, UT 84112
USA")

(setq file-header-user-telephone "+1 801 581 5254")

(setq file-header-user-FAX "+1 801 581 4148")

(setq file-header-user-URL "http://www.math.utah.edu/~beebe")
```

Once this is installed in the '.emacs' file, GNU Emacs will find it every time it starts up.

If the electronic-mail address constructed from the Emacs `user-login-name` and `system-name` functions is not suitable, you can provide an alternative one like this:

```
(setq file-header-user-email "beebe@math.utah.edu")
```

In any of the following situations, you should set `file-header-user-email` in your startup '.emacs' file.

- You work on multiple machines, but prefer to have only one public electronic-mail address.
- At some sites, `system-name` does not return a fully-qualified Internet host name, so the default address constructed by `file-header-email` is unusable outside your local installation.
- Your site is not on the Internet, but you can receive electronic mail via some other network.

The version number is left empty; you can manually insert an appropriate one, perhaps 1.00, or if you are just starting, 0.00.

The checksum and keywords entries are also left empty. There is no point in inserting a checksum until you are ready to save the file, and the keywords have to be supplied by a human.

Now suppose you've just edited a file with such a file header, and you would like to update the header to reflect the changes, and then save the file. All you need to type is `M-x update-file-header-and-save`, and with Emacs' normal command completion, you can probably hit the tab key after the `f` in `file`.

The function `update-file-header-and-save` will update the date and time stamps, the minor version number, the checksum, and save the file.

If the file is a LaTeX file, the date update will also search forward for text that looks something like

```
\\date{29 November 1991 \\
Version 1.01}
```

and change it to the current date and version. That makes it easy to get the version number and revision date printed on the title page.

You can do these updates manually if you like by invoking the functions `update-checksum`, `update-date`, `update-minor-version`, and `update-date-and-minor-version` explicitly.

Major version numbers are rarely changed, and you could easily do the job manually. Nevertheless, for completeness, `update-major-version` is supplied to automate the job.

`update-checksum` will trim trailing whitespace (but leave embedded tabs intact), send the buffer to the '`checksum`' program, and replace it with the output. Don't interrupt it while it is working, or you might lose your file!

The Emacs interface to '`checksum`' has not yet been tested on VAX VMS, so `update-checksum` on that system calls `update-simple-checksum` instead. That function will compute counts of lines, words, and characters and insert them in the checksum value. You could use this if for some reason you don't have `checksum` installed yet. `checksum` should be available from the same place you got '`filehdr.el`'; eventually it will be on dozens of TeX archive machines around the world.

Chapter 7

Simple customization

The GNU Emacs Lisp code in '`filehdr.el`' has been written to make it easy to customize without your having to become a Lisp programmer. Of course, Lisp is so much fun that you might want to do that anyway!

The code contains several large tables stored in Lisp variables:

```
file-header-standard-at-sign-special-cases
file-header-standard-comment-prefixes
file-header-standard-entries
file-header-standard-paired-comment-delimiter-languages
file-header-standard-suffix-and-type
```

These are not intended to be modified by users, as the phrase `-standard-` in their names indicates.

Each of them is a list of lists; the innermost lists contain two or three character strings. Sublists are ordered alphabetically for human readability; the code does not care what order they appear in.

The first of them, `file-header-standard-at-sign-special-cases`, is used to handle those few exceptional file classes that do not permit at-signs, `@`, to be used in comments without special handling. Here is the current value of this variable:

```
(
    ("BibTeX"               " at ")
    ("C-Web"                "@@")
    ("Web"                  "@@")
    ("Web-change"           "@@")
)
```

This means that when a header for a file in class '`BibTeX`' is created, at-signs should be replaced by the string '` at `'. For the other classes, at-signs must be doubled.

The second variable, `file-header-standard-comment-prefixes`, has a very long value, so we show only a portion here:

```
(
    ("Adobe-Font-Metric"    "Comment ")
    ("AmSTeX"               "%%% ")
    ("Awk"                  "### ")
...
    ("Web-change"           "%%% ")
    ("Yacc"                 "")
)
```

This means that in an Adobe Font Metric file, comments must begin a line with the string '`Comment `'. For `awk` files, a triple sharp sign and a space will begin all file header lines. `yacc` file headers have no comment prefix at all.

The third variable, `file-header-standard-entries`, contains pairs of entry names and functions to supply values for them. It looks something like this:

```
(
 ("author"       file-header-author)
 ("version"      file-header-version)
 ("date"         file-header-date)
 ("time"         file-header-time)
 ("filename"     file-header-filename)
 ("address"      file-header-address)
 ("telephone"    file-header-telephone)
 ("FAX"          file-header-FAX)
 ("URL"          file-header-URL)
 ("checksum"     file-header-checksum)
 ("email"        file-header-email)
 ("codetable"    file-header-codetable)
 ("keywords"     file-header-keywords)
 ("supported"    file-header-supported)
 ("docstring"    file-header-docstring)
)
```

The file header is created by processing these entry names in order.

The fourth variable, with the name `file-header-standard-paired-comment-delimiter-languages`, is a little more complex. Its classes cover

languages that use distinct starting and ending comment strings, instead of having comments that terminate at end of line. For each class name, its list entries contain two strings, one for the comment start, and one for the comment end. To help make them stand out better, the strings are often stretched to 72 characters in length:

```
(
    ("C"
     (concat "/*"  (make-string 70 ?\*) "\n")
     (concat (make-string 70 ?\*) "*/\n"))

    ("Font-Property-List"
      (concat "(COMMENT "(make-string 63 ?\*) "\n")
      (concat (make-string 71 ?\*) ")\n"))
...
    ("Scribe"
      "@Begin{Comment}\n"
      "@End{Comment}\n")
...
    ("Yacc"
     (concat " /*" (make-string 69 ?\*) "\n")
     (concat " " (make-string 69 ?\*) "*/\n"))
    )
)
```

To avoid the need for long constant strings in the code, several of them are generated dynamically by the Lisp concatenation operator, `concat`.

Class names in this variable do *not* include the phrase `-file` that appears in the file header; that suffix is supplied automatically by the Emacs functions.

The last variable, `file-header-standard-suffix-and-type`, is the biggest of them all. It relates file extensions to file classes. This indirection was chosen because there are often several file extensions belonging to a single class. Its value looks something like this:

```
(
    ("1"            "Troff-man")
    ("1l"           "Troff-man")
    ("2"            "Troff-man")
...
    ("afm"          "Adobe-Font-Metric")
...
    ("web"          "Web")
```

```
    ("y"            "Yacc")
    ("yacc"         "Yacc")
)
```

Observe that the extensions do *not* include a leading period.

The list of extensions was constructed by going through some large UNIX file systems (several hundred thousand files) to produce a set of unique file extensions, and then augmenting the list by hand based on the author's personal experience on several other operating systems. The resulting list has about 150 file extensions, and 85 file classes. If a file extension is unrecognized, it is assigned the class name `UNKNOWN`.

Here now is how you can customize the behavior of `make-file-header`. For each Lisp variable with the phrase `-standard-`, there is a corresponding one with the phrase `-extra-` instead. These new variables are intended for user customization; you can initialize them in your startup '`.emacs`' file, and they will automatically be added to the standard ones at run time.

Here is a set of sample customizations:

```
(setq file-header-extra-at-sign-special-cases
      '(
        ("Foo-Bar"      " <<<AT>>> ")
        ))

(setq file-header-extra-comment-prefixes
      '(
        ("Foo-Bar"      "!FB!")
        ))

(setq file-header-extra-entries
      '(
        ("copyright"    file-header-copyright)
        ))

(setq file-header-extra-suffix-and-type
      '(
        ("foobar"       "Foo-Bar")
       ))

(setq file-header-extra-paired-comment-delimiter-languages
      '(
        ("Foo-Bar"
         (concat "/#"  (make-string 70 ?\#) "\n")
         (concat (make-string 70 ?\#) "#/\n"))
```

```
))
```

These would define a new file class `Foo-Bar` attached to files with extension `.foobar`, for which comments are delimited by `/# ... #/`, and by `!` to end-of-line. The file header body lines would all begin with `!FB!`.

The Lisp form `(setq var value)` assigns `value` to the variable `var`; most other programming languages would write this as `var = value`.

The extra values set in these variables are appended to the end of the standard ones, so they can augment, *but not replace*, the standard values. This design choice was made intentionally to encourage *standardization* of the file headers. If you need to do something differently, you'll have to learn some Lisp, and look in the next chapter.

You can test your additions by visiting files with the new extensions, and then running `M-x make-file-header`.

You can test the entire collection of code by typing `M-x test-file-header`. This takes a while, but is thorough: it will create file headers in a temporary editor buffer for every file extension defined in the two lists `file-header-standard-suffix-and-type` and `file-header-extra-suffix-and-type`.

To see the settings of the variables named `file-header-standard-xxx` and `file-header-extra-xxx`, type `M-x show-file-header-variables`. The results will appear in a temporary buffer.

Prior to version 19 (released in early summer of 1993), GNU Emacs did not provide the time zone, but on UNIX systems, it can be obtained from the output of the `date` command. Since this takes a few seconds to run as a subprocess, the result is saved in a global variable, `file-header-timezone-string`. Subsequent file headers will be produced much more rapidly. With Version 19 or later, this delay is eliminated.

If you find the delay on the first use objectionable, you can set the time zone in your '`.emacs`' file:

```
(setq file-header-timezone-string "MST")
```

This practice is not recommended, since you'll have to change it twice a year, or if you work in a different time zone.

Chapter 8

Advanced customization

What do you do if you want to insert additional fields in all new file headers? You have to do some Lisp programming to add to the functions in '`filehdr.el`'. *Under no circumstances should you modify* '`filehdr.el`' *itself!* That is the sole prerogative of its original author. You can freely copy code from it, but put that code in a file with a different name.

If you are a real Lisp wizard, you can just read the code in '`filehdr.el`', and write whatever new code you want. On the other hand, if you were such a wizard, you'd probably "read the code instead of this documentation."

The most likely function you'll want to modify is `make-file-header`. Here is what its body looks like:

```
(file-header-comment-block-begin)
(file-header-entry)
(mapcar '(lambda (entry)
           (file-header-key (car entry) (nth 1 entry)))
        (append file-header-standard-entries
                file-header-extra-entries))
(file-header-exit)
(file-header-comment-block-end)
```

Each of these lines is a Lisp function call; the function name is the first one in each parenthesized list. Each function supplies part of the standard file header.

The first and last function calls provide a full line comment start and end, if the file class requires it.

The `file-header-entry` and `file-header-exit` functions supply the class name tag and the final closing brace. That is, they generate something like this:

```
%%%  @LaTeX-file{
%%%  }
```

The individual file attributes are then supplied by calls to the generic function `file-header-key`, which is given the attribute name as its first argument, and the name of a function to call to generate a string for the attribute's initial value. The returned string may span multiple lines; it will be neatly formatted and properly indented by a service function called inside `file-header-key`.

The Lisp `mapcar` function called in the body of `make-file-header` applies its second argument, here an anonymous `lambda` function, to each element of the list supplied as its third argument. The keywords that are inserted are determined by the entries in the lists `file-header-standard-entries` and `file-header-extra-entries`, which are appended into one big list.

Here is a simple example of one of these initial value-returning functions:

```
(defun file-header-codetable ()
  "Return as a string the default codetable value."
  "ISO/ASCII"
  )
```

If you want to add a new file header attribute entry, you need to add an entry to `file-header-extra-entries`, and write a function to return an appropriate initial value.

This is best illustrated by a real example—the addition of a copyright attribute in the file header.

First we insert the lines

```
(setq file-header-extra-entries
      '(
        ("copyright" file-header-copyright)
        ))
```

in the '`.emacs`' file.

Next, we write the function to return the initial value:

```
(defun file-header-copyright ()
  "Return as a string the default copyright value."
  "None.  This file is PUBLIC DOMAIN."
)
```

That is all there is to it. To test the new code, you can compile it inside Emacs in Emacs-Lisp editing mode by typing `ESC C-x` with the cursor

inside the function, and then run it by name from the minibuffer: `ESC ESC (file-header-copyright)`.

When you run `make-file-header`, it should now produce an attribute entry like

```
%%%     copyright     = "None.  This file is PUBLIC DOMAIN.",
```

When everything is working, save the new Emacs Lisp file, and run `M-x byte-compile-file` on it. You can then load it interactively with `M-x load-file`, or better, automatically at Emacs start-up time by adding the line

```
(load "myfilhdr" t t nil)
```

assuming you called the modified file '`myfilhdr.el`'.

If the code in '`myfilhdr.el`' is short, you can keep it in your '`.emacs`' instead, and altogether avoid the need for a separate file and the byte compilation and `load` command. Compilation is only useful for speeding up the loading of large files of Emacs Lisp code.

You probably will not have to do any more than this, unless you add a new attribute that must be updated each time the function `update-file-header-and-save` is invoked. In such a case, you'll have to study its body, and the functions it calls, to make the necessary modifications.

Chapter 9

Bug reporting

Bug reports, and comments, are actively solicited. Electronic mail to the author is most convenient, but postal mail, preferably accompanied by machine-readable material on Apple Macintosh or IBM PC floppy disks, are also acceptable. Shorter communications via FAX are also possible. Here are the necessary addresses and telephone numbers:

Nelson H. F. Beebe
Center for Scientific Computing
Department of Mathematics
University of Utah
Salt Lake City, UT 84112
USA
Tel: +1 801 581 5254
FAX: +1 801 581 4148
Email: beebe@math.utah.edu
URL: http://www.math.utah.edu/~beebe

Bibliography

[1] Nelson H. F. Beebe. From the President. *TUGboat*, 11(4):485–487, November 1990.

[2] Leslie Lamport. LaTeX—*A Document Preparation System—User's Guide and Reference Manual*. Addison-Wesley, 1985.

[3] Mark Lottor. Internet domain system. *Communications of the Association for Computing Machinery*, 34(11):21–22, November 1991. This letter reports that the ZONE program at the Network Information Systems Center at SRI International in July 1991 found approximately 535,000 Internet hosts in 16,000 domains. The 10 largest domains were EDU (educational)–206,000, COM (commercial)–144,000, GOV (government)—36,000, MIL (military) 26,000, AU (Australia)–22,000, DE (Germany)—21,000, CA (Canada)–19,000, ORG (organizations)—15,000, SE (Sweden)—12,000, and CH (Switzerland)—10,000.

[4] Unilogic, Ltd. *Scribe Document Production System User Manual*, April 1984.

Concept index

A

B

C

D

E

F

G

H

I

J

K

L

N

O

P

S

T

U

V

W

Function index

U

Person index

Program index

Variable index

F