Computing Center Memo 815 *August 1987 Revised: August 1988*

Using LaTeX on MTS

by Leslie Lamport, modified by U-M TeX support staff
University of Michigan Computing Center
Ann Arbor, Michigan

Contents

1 Getting Started **3**
1.1 Running a Sample File 3
1.2 Preparing and Running LaTeX on Your Own Files 4

2 Carrying On **5**
2.1 LaTeX Files on MTS 5
2.2 Document Styles 6
2.2.1 The `rackham` Style 6
2.2.2 The `proc` Style Option 6
2.2.3 The `bezier` Style Option 7
2.2.4 The `ifthen` Style Option 7
2.2.5 Letters 8
2.3 Where the Files Are 8
2.4 Running `lablst.tex` and `idx.tex` 8
2.5 Differences from the Manual 9
2.6 Using BibTeX 9
2.7 Using SliTeX 11
2.8 Fonts 11
2.9 Special Versions 11

3 Errata and Additions to the Manual **12**

A Bibliography-style Hacking **16**

A.1 General description 17
A.2 Commands 18
A.3 The built-in functions 20
A.4 Name formatting 23

Using LaTeX on MTS

LaTeX runs on a variety of computers at many different sites. This document tells you how to use LaTeX using MTS on the UM and UB systems at The University of Michigan. It is not about LaTeX itself, which is described by the manual—*LaTeX: A Document Preparation System*, published by Addison-Wesley, available at fine book stores everywhere.

If you have a question that you can't answer by reading the manual and this document, ask the TeX Support Staff. They should also be informed of any possible LaTeX bugs or undocumented anomalies. They can be reached by $MESSAGE, send to TeX_Support_Staff.

1 Getting Started

1.1 Running a Sample File

Before preparing your own documents, you may want to get acquainted with LaTeX by running it on a sample input file. First make your own copy of the file `sample.tex` by typing the following MTS command:

```
$duplicate tex:sample.tex sample.tex
```

(This and all other MTS commands are entered by pressing the *return* key.) A copy of the file `sample.tex` is now stored on disk on your CCID; you can edit it just like any other file. If you destroy or mess up your copy, typing the above command again gets you a fresh one.

Next, run LaTeX on the file `sample.tex` by typing:

```
$create sample.aux
$create sample.dvi
$create sample.xer
$run *latex par=sample
```

When LaTeX has finished, it will have produced the file `sample.dvi`. You can print this file by typing the commands

```
    $run *Dvixer par=sample.dvi
    $run *Pagepr scards=sample.xer
```

The output will be produced on a Xerox 9700 page printer, located either at CNTR, UNYN, or NUBS. Type `$HELP 9700` for more information on page printer use.

After it has been printed, you can delete all the sample files by typing

```
     $destroy sample.?
```

and answering yes to the prompt for confirmation.

1.2 Preparing and Running LaTeX on Your Own Files

You must use a text editor to prepare an input file for LaTeX. The User Guide *Introduction to the MTS File Editor–Getting it Right* describes the text editor available on MTS. You can also prepare files offline on a microcomputer and telecopy them to MTS. Currently, the supported programs are MacWrite for the Macintosh, and Microsoft Word for both the Macintosh and for IBM PCs and PC-compatibles. The easiest way to start learning about LaTeX is by examining the file `small.tex` with your text editor. You can obtain your own copy of this file, on your CCID, by typing the command

```
     $duplicate tex:small.tex small.tex
```

After you have prepared your file, whose name should have the extension `tex`, you must run it through LaTeX and print the output. Follow the instructions in Section 1.1, except substitute the first name of your file for “`sample`”. Remember to save disk charges by deleting the `dvi` and `xer` file after you are all done with that document.

Since LaTeX is not a text editor, some other software must be used to enter the text of a LaTeX document. Some text editors use control codes that may not be visible but could cause bad characters to appear in the input file and generate the

```
! Text line contains an invalid character.
```

error. With MTS, you can use the MTS Editor to edit documents without entering any special control codes or non-printing characters. You can also enter text offline using any of the microcomputer word processing programs supported by the U-M Computing Center and telecopy it to MTS to use as LaTeX input without special character problems. Word documents must be saved as plain ASCII files in this case. In general, other editors or word processors that produce ASCII output will not cause character problems, but it is wise to test a sample file before committing yourself to the use of a non-supported system.

If you want to stop LaTeX in the middle of its execution, perhaps because it is printing a seemingly unending string of uninformative error messages, hit the *Break* or *Attn* key, or type *Control-e* (press *e* while holding down the key labeled *CTRL*). This will make LaTeX stop and return you to MTS command mode. If you change your mind you can type `$Restart` to continue.

2 Carrying On

2.1 LaTeX Files on MTS

The only special concern in using LaTeX that is caused by the way files are handled by TeX on MTS arises because when a TeX program starts to write a file, it destroys the previous version of that file. Thus, if an error forces you to stop LaTeX prematurely (by typing *Break* or *Attn*), then the files that LaTeX was writing are incomplete, and the previous complete versions have been destroyed. You probably don't care about the output on the `dvi` file, but, if you are making a table of contents or using cross-referencing commands, then LaTeX also writes one or more *auxiliary files* that it reads the next time it processes the same input file. If the auxiliary files are incomplete because LaTeX was stopped before reaching the end of its input file, then the table of contents and cross-references will be incorrect the next time LaTeX is run on the same input file. You will have to run LaTeX a second time to get them right. If you want to avoid having to run LaTeX twice after making an error—for example, if your input is very long—then you should save copies of these auxiliary files before running LaTeX. An input file named `myfile.tex` and all the auxiliary files produced by LaTeX from it are included in the MTS file pattern `myfile.?`. Use the MTS `$duplicate` command to save copies of these files.

The second possible problem in using LaTeX on MTS involves the way that LaTeX on MTS searches public and user CCIDs for the files that LaTeX reads. In addition to reading your document's text input file, LaTeX also reads the files specified by `\input` and `\include` commands. LaTeX will look for these files first on your CCID and next on the CCID TEX. It will have no problem finding the correct files if you follow two simple rules:

1. To avoid confusion, give your files unique names, not the same file names as the public files that reside on the TEX CCID.
2. If you want to use an input file from another CCID, specify the name of the file fully in the form `ccid:filename`.

2.2 Document Styles

There are four document styles and style options available on MTS that are not described in the manual: the `rackham` document style for dissertations submitted to U-M's Rackham School of Graduate Studies; the `proc` style option for making camera-ready copy for conference proceedings, the `bezier` option for drawing curves, and the `ifthen` option for implementing **if-then-else** and **while-do** control structures. They are described below.

2.2.1 The `rackham` Style

The `rackham` style is a separate document style. It produces output in the format required by the Rackham School of Graduate Studies for doctoral dissertations. It was written by Doug Maus for CAEN (the Computer-Aided Engineering Network) and is modelled after the TEX dissertation package documented in CC Memo 814, "TEX and LATEX Macros for Formatting a Rackham Dissertation." The revised version of Memo 814 describes how to use it. The CAEN documentation for it is available at Dollar Bill Copying.

2.2.2 The `proc` Style Option

The `proc` option is used with the `article` document style. It produces two-column output for ACM and IEEE conference proceedings. The command `\copyrightspace` makes the blank space at the bottom of the first column of the first page, where the proceedings editor will insert a copyright notice. This command works by producing a blank footnote, so it is placed in the text of the first column. It must go after any `\footnote` command that generates a footnote in that column.

LATEX automatically numbers the output pages. It's a good idea to identify the paper on each page of output. Placing the command

```
\markright{Jones---Foo}
```

in the preamble (before the `\begin{document}` command) prints "Jones—Foo" at the bottom of each page.

2.2.3 The `bezier` Style Option

This option defines a single command, `\bezier`, that draws a curved line in a `picture` environment. Let P_i be the point with coordinates (x_i, y_i), for $i = 1$, 2, and 3. The command

```
\bezier{n}(x1,y1)(x2,y2)(x3,y3)
```

draws n points on the quadratic Bezier spline determined by the three points P_1, P_2, and P_3. The locus of points on this spline is a parabolic arc from P_1 to P_3 having the line P_1P_2 tangent to it at P_1 and the line P_2P_3 tangent to it at P_3. Note that P_2 is *not* on this arc unless P_1, P_2, and P_3 are colinear, in which case the arc is a straight line. Bezier splines are useful because it's easy to join two of them together smoothly by giving them the same tangent line where they meet.

It takes roughly 75 points per inch to form a solid line, depending upon the line thickness. See Section C.13.3 of the LaTeX manual for commands to specify line thickness in a `picture` environment. This command is *very* slow, and TeX has enough memory to hold only about 1000 points plus a page of text. (Remember that TeX keeps the current page plus all as yet unprinted figures in memory.) So, the `bezier` command should be used for only a small number of small curves.

2.2.4 The `ifthen` Style Option

This option provides two programming language features that are useful only for people who already know how to program. It defines the two commands

```
\ifthenelse{test}{then clause}{else clause}
\whiledo{test}{do clause}
```

that implement the following two Pascal language structures

```
if test then then clause
        else else clause
while test do do clause
```

The *then*, *else*, and *do* clauses are ordinary LaTeX input; *test* is one of the following:

- A relation between two numbers formed with `<`, `>`, or `=`; for example, `\value{page}>3`.

- `\equal{`*string1*`}{`*string2*`}`, which evaluates to *true* if *string1* and *string2* are the same strings of characters after all commands have been replaced by their definitions. (Upper- and lowercase letters are unequal.)
- A logical combination of the above two kinds of tests using the operators `\or`, `\and`, and `\not` and the parentheses `\(` and `\)`—for example:

  ```
  \not \( \value{section} = 1
         \and  \equal{Jones}{\myname} \)
  ```

These commands, together with `\renewcommand` and the commands of Section C.7.4 for manipulating counters, open up a whole new world of hacking.

2.2.5 Letters

The `letters` document style, described in the manual, should be used for generating personal letters. At the moment there is no special style available for generating letters to be copied onto U-M letterhead. There are no features for making letters other than those described in the manual. However, suggestions will be accepted for such options.

2.3 Where the Files Are

All LaTeX files mentioned in the LaTeX manual, including the `sty` and `doc` files, are on the CCID TEX. The `tfm` files used by TeX are on TEX, but the pixel files used by the Xerox 9700 are not accessible online.

2.4 Running `lablst.tex` and `idx.tex`

A list of labels and citations in an input file is printed by running LaTeX on the input file `lablst.tex`, which is done by typing

```
$Run *latex par=lablst
```

LaTeX will then ask for the name of the input file, which should be typed without an extension, and for the name of the main document style (e.g., `article`), used by that file.

The index entries on an `idx` file are printed by running LaTeX on the file `idx.tex`, which is done by typing

```
$Run *latex par=idx
```

LaTeX will ask for the name of the `idx` file, which is typed without an extension.

2.5 Differences from the Manual

All LaTeX features described in the manual are provided by the implementation at U-M on MTS. However, SliTeX is unavailable.

Disks and circles are available in sizes from 1 to 15 points, in 1-point increments, and 1/4 circle or oval 'corners' in sizes that produce 4, 8, 12, 16, 20, 24, 28, 32, 36 and 40-point circles.

2.6 Using BibTeX

BibTeX is a program for compiling a reference list for a document from a bibliographic database. The current MTS version of BibTeX, Version .98i, corresponds to our current LaTeX version 2.09, but was not installed when LaTeX was originally made available.

If you are using LaTeX and BibTeX on a variety of computing systems, pay attention to the version numbers involved, since .98i .bst files are incompatible with .99a and vice-versa. PCTeX uses .98i. BibTeX's author, Oren Patashnik, has announced that he will freeze the program at v.1.00, after which there will be bug fixes only, so the MTS version will probably be upgraded when 1.00 is available.

BibTeX is run by typing

```
$Run *BIBTEX par=myfile
```

where `myfile.tex` is the name of your LaTeX input file. This reads the file `myfile.aux`, which was generated when you ran LaTeX on `myfile.tex`, and produces the file `myfile.bbl`. BibTeX should be run from the CCID containing `myfile.tex` (which should be the same CCID from which LaTeX was run on that file).

If the `bib` file is not on the same CCID as the LaTeX input file—for example, if you're using someone else's `bib` file—then you must include a CCID as part of the file name specified by the `\bibliography` command. For example, the LaTeX command

```
\bibliography{1XYZ:gnus}
```

specifies the file `gnus.bib` kept by Jones on his `1XYZ` CCID.

There is now no formal provision for sharing bibliographic database information, nor are there programs to assist in making your own `bib` files. Suggestions for forming one or more common `bib` files are welcome.

In addition to the bibliography styles described in the manual, there is a `ieeetr` style that formats entries in the style of the IEEE transactions. Users can also customize styles to their particular requirements. Details on how to customize styles are provided in Appendix A of this writeup.

In addition to the usual three-letter abbreviations for the months, the following abbreviations are defined by the bibliography styles:

`acmcs` *ACM Computing Surveys*

`acta` *Acta Informatica*

`cacm` *Communications of the ACM*

`ibmjrd` *IBM Journal of Research and Development*

`ibmsj` *IBM Systems Journal*

`ieeese` *IEEE Transactions on Software Engineering*

`ieeetc` *IEEE Transactions on Computers*

`ieeetcad` *IEEE Transactions on Computer-Aided Design of Integrated Circuits*

`ipl` *Information Processing Letters*

`jacm` *Journal of the ACM*

`jcss` *Journal of Computer and System Sciences*

`scp` *Science of Computer Programming*

`sicomp` *SIAM Journal on Computing*

`tocs` *ACM Transactions on Computer Systems*

`tods` *ACM Transactions on Database Systems*

`tog` *ACM Transactions on Graphics*

`toms` *ACM Transactions on Mathematical Software*

`toois` *ACM Transactions on Office Information Systems*

`toplas` *ACM Transactions on Programming Languages and Systems*

size	default (10pt)	11pt option	12pt option
`\tiny`	5pt	6pt	6pt
`\scriptsize`	7pt	8pt	8pt
`\normalsize`	10pt	11pt	12pt
`\large`	12pt	12pt	14pt
`\Large`	14pt	14pt	17pt
`\LARGE`	17pt	17pt	20pt
`\huge`	20pt	20pt	25pt
`\Huge`	25pt	25pt	25pt

Table 1: Type sizes for LaTeX size-changing commands.

`tcs` *Theoretical Computer Science*

Note: All styles should share the same set of abbreviations.

2.7 Using SliTeX

SliTeX is a version of LaTeX for making slides. This is presently unavailable.

2.8 Fonts

Almost all the symbols available on our fonts can be generated by ordinary LaTeX commands. However, there are type faces and sizes not obtainable by LaTeX's size-changing commands with the ordinary document styles. Consult CC Memo 811, "TeX Fonts Available on the Xerox 9700," to find the TeX name for such a font.

Tables 1 and 2 allow you to determine if the font for a type style at a particular size is preloaded, loaded on demand, or unavailable. Table 1 tells you what size of type is used for each LaTeX type-size command in the various document-style options. For example, with the `12pt` option, the `\large` declaration causes LaTeX to use 14pt type. Table 2 tells, for every type size, to which class of fonts each type style belongs. For example, in 14pt type, `\bf` uses a preloaded font and the other five type-style commands use load-on-demand fonts. Roman (`\rm`) and math italic (`\mit`) fonts are all preloaded; the `\em` declaration uses either italic (`\it`) or roman.

2.9 Special Versions

No foreign-language or other special versions of LaTeX are currently available on MTS.

	`\it`	`\bf`	`\sl`	`\sf`	`\sc`	`\tt`
5pt	X	D	X	X	X	X
6pt	X	D	X	X	X	X
7pt	P	D	D	D	X	D
8pt	P	D	D	D	X	D
9pt	P	P	D	D	X	P
10pt	P	P	P	P	D	P
11pt	P	P	P	P	D	P
12pt	P	P	P	P	D	P
14pt	D	P	D	D	D	D
17pt	D	P	D	D	X	D
20pt	X	X	X	D	X	X
25pt	X	X	X	D	X	X

Table 2: Font classes: P = preloaded, D = loaded on demand, X = unavailable.

3 Errata and Additions to the Manual

These are all the errors and omissions to the manual, LaTeX: *A Document Preparation System* reported by 16 January 1986.

page xiii

Add Mike Urban to the list of people thanked in the third paragraph.

page 2

In the first paragraph of Section 1.1, replace the three instances of `sample.tex` by `small.tex`.

page 15, line 18

Replace "thay" by "that".

Section 3.3.5, page 49

The name of the environment is `eqnarray`. There are two instances on this page of the incorrect name "`eqnarry`" that should be changed.

page 52, last line

Replace "instead of $\int\int ydxdy$" by "instead of $\int\int zdxdy$".

pages 55–58, 124, and 173–174

Commands that define or redefine a command or environment, such as the `\newcommand` and `\renewenvironment` commands, should not be nested within one another. Doing so may result in the following TeX error:

```
! Illegal parameter number in definition of ... .
```

page 60, line -7

Replace "one of these environments" by "a `figure` or `table` environment".

page 75, line 15

Change `\thebibliography` to `\bibliography`.

page 88, line 27

Change "entire the paragraph"to "the entire paragraph".

page 89, line 16

Change "page-breaking" to "line-breaking".

page 95, line 5

Replace "`{.01in}`" by "`{1.01in}`".

page 96

In the penultimate paragraph of Section 5.4.2, replace

> The `\vfill` command is an abbreviation for `\vspace{\fill}`.

with the following:

> The `\vfill` command is equivalent to a blank line followed by `\vspace{\fill}`.

page 98, line 6

Remove an "i" from "directiion".

page 102

In Figure 5.1, replace the two occurrences of "-1.8" by "-1.8".

page 116, line 20

Change "fourteen-point Plus Roman" to "twelve-point Plus Roman".

page 118, line -10

Change this line to:

```
... (myfile.tex [1] [2] [3] (part1.tex [4] [5]) (part2.tex [6] [7]
```

page 142

Add the following near the bottom of the page, just above the **Titles** heading.

> If an `author` or `editor` field has more names than you want to type, just end the list of names with `and others`; the standard styles convert this to the conventional *et al.*

page 160, top line

Change "(Section 5.6)" to "(Section 5.3)".

page 160, line -9

Change the description of the `openbib` style option to: "Causes the bibliography (Section 4.3) to be formatted in open style. (See van Leunen [7].)"

page 168, line -5

Replace "printed as \␣" by "printed as ␣".

page 169, line -5

Change "first and third rows" to "first and third columns".

page 170

Add the following paragraph after line 5:

> An overfull `\hbox` warning occurs if a formula extends beyond the prevailing margins. However, if the formula does lie within the margins, no warning is generated even if it extends far enough to overprint the equation number.

page 176

Change the four lines immediately following the heading for Section C.8.1 to:

```
\begin{figure}[loc]    body   \end{figure}
\begin{figure*}[loc]   body   \end{figure*}
\begin{table}[loc]     body   \end{table}
\begin{table*}[loc]    body   \end{table*}
```

page 185, line −5

Replace “suppresses command” by “command suppresses”.

page 187, lines 15–17

The sentence “It also writes *bib_files*. . . ” is redundant and can be eliminated.

page 191, line 13

The word “paragraph” misspelled.

page 191, line 14

Remove the space between `\begin` and `{sloppypar}`. (This is for consistency only; LaTeX ignores the space.)

page 199

Add the following sentence to the last paragraph on the page:

> Words typeset in typewriter style or in two different styles are not hyphenated except where permitted by `\-` commands.

(This is a change to LaTeX made on 18 December 1985.) Also, add the following index entry citations to this page: "`\-`", "hyphenation, suppressed", and "typewriter type style, no hyphenation in".

page 217

Add the subentry "`openbib`, 160" to the index entry "document-style option".

page 228

In index entry for `\multicolumn`, change "194" to "184".

page 229

Add the index entry "`openbib` document-style option, 160".

page 241

Add page 47 to the index entry for "van Leunen, Mary-Claire".

Tear-Out Command Sheet

In the first column, sixth line after "Sentences and Paragraphs" heading, replace "& `&`" by "& `\&`".

A Bibliography-style Hacking

This section is not intended for ordinary LaTeX users. Together with the standard-style documentation in the LaTeX manual, it should explain how to modify existing style files and to produce new ones. If you're a serious style hacker you should be familiar (in order of importance) with van Leunen [6] for points of style, with Lamport [3] and Knuth [2] for formatting matters, and with *Scribe* [5] for compatibility details. And while you're at it, if you don't read the great little book by Strunk and White [4], you should at least look at its database and reference-list entries to see how BibTeX handles multiple names.

If you find any bugs in the standard styles or if you can't do things you'd like with bibliography-style files, please complain to Oren Patashnik.

A.1 General description

You write bibliography-style files in a postfix stack language. It's not too hard to figure out how by looking at the standard-style documentation, but this description fills in a few details (it will fill in more details if there's a demand for it).

Basically the style file is a program, written in an unnamed language, that tells BibTeX how to format the entries that will go in the reference list (henceforth "the entries" will be "the entry list" or simply "the list", context permitting). This programming language has ten commands, described in the next subsection. These commands manipulate the language's objects: constants, variables, functions, the stack, and the entry list. (Warning: The terminology in this documentation, chosen for ease of explanation, is slightly different from BibTeX's. For example, this documentation's "variables" and "functions" are both "functions" to BibTeX. Keep this in mind when interpreting BibTeX's error messages.)

There are two types of functions: *built-in* ones that BibTeX provides (these are described in Section A.3), and ones you define using either the `MACRO` or `FUNCTION` command. As a style designer, creating or modifying functions using the `FUNCTION` command will be your most time-consuming task (actually, becoming familiar with the references listed above will be, but assume for the moment that's done).

Let's look at a sample function fragment. Suppose you have a string variable named `label` and an integer variable named `lab.width`, and suppose you want to append the character `a` to `label` and increment `lab.width`:

```
...
'label label "a" * :=          % label := label * "a"
'lab.width lab.width #1 + :=   % lab.width := lab.width + 1
...
```

In the first line, `'label` pushes the variable name onto the stack. Next, `label` pushes its value, and `"a"` pushes the string constant `a` onto the stack. Then the built-in function `*` pops the top two strings and pushes their concatenation. Finally, the built-in function `:=` pops the concatenation and variable name and performs the assignment. BibTeX treats the stuff following the `%` as a comment. The second line is similar except that it uses `#1` (with no intervening spaces) to push the integer constant.

The nonnull spacing here is arbitrary: multiple spaces, tabs, or newlines are equivalent to a single one (except that you're probably better off not having blank lines within commands, as explained shortly).

For string constants, absolutely any printing character is legal between two consecutive double quotes, but BibTeX here (and only here) treats upper- and lower-case equivalents as different. Furthermore, spacing *is* relevant within a string constant, and you mustn't split a string constant across lines (that is, the beginning and ending double quotes must be on the same line).

Variable and function names may not begin with a numeral and may not contain any of the ten restricted characters described earlier, but may otherwise contain any printing characters. Also, BibTeX treats upper- and lower-case equivalents as the same.

Integers and strings are the only value types for constants and variables (booleans are simply 0-or-1 integers). There are three kinds of variables:

global variables These are either integer- or string-valued, declared using an `INTEGERS` or `STRINGS` command.

entry variables These are either integer- or string-valued, declared using the `ENTRY` command. Each has a value for each entry on the list (example: a variable `label` might store the label string you'll use for the entry).

fields These are string-valued, read-only variables that store the information from the database file, and whose values are set by the `READ` command. As with entry variables, each has a value for each entry.

A.2 Commands

There are ten style-file commands: Five (`ENTRY`, `FUNCTION`, `INTEGERS`, `MACRO`, and `STRINGS`) declare and define variables and functions, one (`READ`) reads in the database information, and four (`EXECUTE`, `ITERATE`, `REVERSE`, and `SORT`) manipulate the entries and produce output. Although the command names appear here in upper case, BibTeX ignores case differences.

The order restrictions are: There must be exactly one `ENTRY` and one `READ` command; the `ENTRY`, all `MACRO`, and certain (see `call.type$`) `FUNCTION` commands must precede the `READ` command; and the `READ` command must precede the four that manipulate the entries and produce output.

Also it's best (but not essential) to leave at least one blank line between commands and to leave no blank lines within a command; this helps BibTeX recover from any syntax errors you make.

You must enclose each argument of every command in braces. Look at the standard-style documentation for syntactic issues not described here.

`ENTRY` Declares the fields and entry variables. It has three arguments, each a (possibly empty) list of variable names. The three lists are of: fields, integer, and string entry variables. There is an additional string entry variable that BibTeX automatically declares, `sort.key$`, used by the `SORT` command. Each of these variables has a value for each entry on the list.

`EXECUTE` Executes a single function. It has one argument, the function name.

`FUNCTION` Defines a new function. It has two arguments; the first is the function's name and the second is its definition. You must define a function before using it; recursive functions are thus illegal.

`INTEGERS` Declares global integer variables. It has one argument, a list of variable names. You may have any number of these commands, but a variable's declaration must precede its use.

`ITERATE` Executes a single function, once for each entry in the list, in the list's current order (initially the list is in order of occurrence, but the `SORT` command may change this). It has one argument, the function name.

`MACRO` Defines a string macro. It has two arguments; the first is the macro's name, which is treated like any other variable or function name, and the second is its definition, which must be double-quote-delimited. You must have one for each three-letter month abbreviation; in addition, you should have one for common journal names. The user's database may override any definition you define using this command. If you want to define a string the user can't touch, use the `FUNCTION` command, which has a compatible syntax.

`READ` Dredges up from the database file the field values for each entry in the list. It has no arguments. If a database entry doesn't have a value for a field (and probably no database entry will have a value for every field), that field variable is marked as missing for the entry.

`REVERSE` Exactly the same as the `ITERATE` command except that it executes the function in reverse order of the entry list.

`SORT` Sorts the entry list using the values of the string entry variable `sort.key$`. It has no arguments.

`STRINGS` Declares global string variables. It has one argument, a list of variable names. You may have any number of these commands, but a variable's declaration must precede its use.

A.3 The built-in functions

There are currently 32 built-in functions. Every built-in function with a letter in its name ends with a `$`. In what follows, "first", "second", etc. refer to the order popped. A "literal" is an element on the stack, and it will be either an integer value, a string value, a variable or function name, or a special value denoting a missing field. If any popped literal has an incorrect type, BibTeX complains and pushes the integer 0 or the null string, depending on whether the function was supposed to push an integer or string.

`>` Pops the top two (integer) literals, compares them, and pushes the integer 1 if the second is greater than the first, 0 otherwise.

`<` Analogous.

`=` Pops the top two (both integer or both string) literals, compares them, and pushes the integer 1 if they're equal, 0 otherwise.

`+` Pops the top two (integer) literals and pushes their sum.

`-` Pops the top two (integer) literals and pushes their difference (the first subtracted from the second).

`*` Pops the top two (string) literals, concatenates them (in reverse order, that is, the order in which pushed), and pushes the resulting string.

`:=` Pops the top two literals and assigns to the second (which must be a global or entry variable) the value of the first.

`add.period$` Pops the top (string) literal, adds a "`.`" to it if the last non"}" character isn't a "`.`", "`?`", or "`!`", and pushes this resulting string.

`call.type$` Executes the function whose name is the entry type of an entry. For example if an entry is of type `book`, this function executes the `book` function. When given as an argument to the `ITERATE` command, `call.type$` actually produces the output for the entries. For an entry with an unknown type, it executes the function `default.type`. Thus you should define (before the `READ` command) one function for each standard entry type as well as a `default.type` function.

`change.case$` Pops the top two (string) literals; it changes the case of the second according to the specifications of the first, as follows. (Note: The word "letters" in the next sentence refers only to those at brace-level 0, the top-most brace level; no other characters are changed.) If the first literal is the string `ul`, it converts all letters to lower case except the very first character in the string, which it converts to upper case; if it's the string `uu`, it converts all letters to upper case; and if it's the string `ll` or `lu`, it converts analogously. It then pushes this resulting string. (Note: It ignores case differences in the specification string; for example, the strings `Ul` and `ul` are equivalent for the purposes of this built-in function.)

`chr.to.int$` Pops the top (string) literal, makes sure it's a single character, converts it to the corresponding ASCII integer, and pushes this integer.

`cite$` Pushes the string that was the `\cite`-command argument for this entry.

`duplicate$` Pops the top literal from the stack and pushes two copies of it.

`format.name$` Pops the top three literals (they are a string, an integer, and a string literal, in that order). The last string literal represents a name list (each name corresponding to a person), the integer literal specifies which name to pick from this list, and the first string literal specifies how to format this name, as explained in the next subsection. Finally, this function pushes the formatted name.

`if$` Pops the top three literals (they are two function literals and an integer literal, in that order); if the integer is greater than 0, it executes the second literal, else it executes the first.

`int.to.chr$` Pops the top (integer) literal, interpreted as the ASCII integer value of a single character, converts it to the corresponding single-character string, and pushes this string.

`int.to.str$` Pops the top (integer) literal, converts it to its (unique) string equivalent, and pushes this string.

`missing$` Pops the top literal and pushes the integer 1 if it's a missing field, 0 otherwise.

`newline$` Writes onto the `bbl` file what's accumulated in the output buffer. It writes a blank line if and only if the output buffer is empty. Since `write$` does reasonable line breaking, you should use this function only when you want a blank line or an explicit line break.

`num.names$` Pops the top (string) literal and pushes the number of names the string represents—one plus the number of occurrences of the substring "and" (ignoring case differences) surrounded by nonnull white-space at the top brace level.

`pop$` Pops the top of the stack but doesn't print it; this best gets rid of an unwanted stack literal.

`purify$` Pops the top (string) literal, removes from it nonalphanumeric characters except for white-space characters (which get converted to spaces), and pushes the resulting string.

`quote$` Pushes the string consisting of the double-quote character.

`skip$` Is a no-op.

`stack$` Pops and prints the whole stack; it's meant to be used for style designers while debugging.

`substring$` Pops the top three literals (they are the two integers literals *len* and *start* and a string literal, in that order). It pushes the substring of the (at most) *len* consecutive characters starting at the *start*th character (assuming 1-based indexing) if *start* is positive, and ending at the $-$*start*th character from the end if *start* is negative (where the first character from the end is the last character).

`swap$` Swaps the top two literals on the stack.

`top$` Pops and prints the top of the stack on the terminal and log file. It's useful for giving the user warning and error messages.

`type$` Pushes the current entry's type (book, article, etc.), but pushes the null string if the type is either unknown or undefined.

`while$` Pops the top two (function) literals, and keeps executing the second as long as the (integer) literal left on the stack by executing the first is greater than 0.

`width$` Pops the top (string) literal and pushes the integer that represents its width in some relative units (currently, hundredths of a point, as specified by the July 1984 version of the amr10 font; the only white-space character with nonzero width is the space). This is meant to be used for comparing widths of label strings.

`write$` Pops the top (string) literal and writes it on the output buffer (which will result in stuff being written onto the `bbl` file if the buffer fills up).

Note that the built-in functions `while$` and `if$` require two function literals on the stack. You get them there either by immediately preceding the name of a function by a single quote, or, if you don't feel like defining a new function with the `FUNCTION` command, by simply giving its definition (that is, giving what would be the second argument to the `FUNCTION` command, including the surrounding braces). For example the following function fragment appends the character `a` if the string variable `label` is nonnull:

```
...
label "" =
  'skip$
  { 'label label "a" * := }
if$
...
```

A function whose name you quote needn't be built in (unlike `skip$` above)—it may be one you've defined earlier.

A.4 Name formatting

What's in a name? The LaTeX manual [3] pretty much describes this. Each name consists of four parts: First, von, Last, and Jr; each consists of a list of name-tokens, and any list but Last may be empty for a nonnull name. This subsection describes the format string you must supply to the built-in function `format.name$`.

Suppose a database file has the field

```
AUTHOR="Charles Louis Xavier Joseph de la Vall\'ee Poussin"
```

in an entry [1], and suppose you want this formatted "last name comma abbreviated first name". If you use the format string

```
"{vv~}{ll}{, jj}{, f}?"
```

BibTeX will produce

```
de~la~Vall\'ee~Poussin, C.~L.~X.~J?
```

as the formatted string.

Let's look at this example in detail. There are four brace-level 1 *pieces* to this format string, one for each part of a name. If the corresponding part of a name isn't present (the Jr part for this name), everything in that piece is ignored. Anything at brace-level 0 is output verbatim (the presumed typo "?" for this name), but you probably won't use this feature much.

Within each piece a double letter tells BibTeX to use whole tokens, and a single letter to abbreviate them (these letters must be at brace-level 1); everything else within the piece is used verbatim (well, almost everything—read on). BibTeX puts default strings *between* tokens of a name part: For whole tokens it uses a tie and for abbreviated ones it uses a period followed by a tie. However it doesn't use this default string after the last token in a list; hence there's no period following the "J" for our example. You should have used

```
"{vv~}{ll}{, jj}{, f.}"
```

to get BibTeX to produce the same formatted string but with the question mark replaced by a period. Note that the period should go inside the First-name piece, rather than where the question mark was, in case a name has no First part.

If you want to override BibTeX's default between-token strings, you must explicity specify a string. For example, suppose you want a label to contain the first letter from each token in the von and Last parts, with no spaces; you should use the format string

```
"{v{}}{l{}}"
```

so that BibTeX will produce

```
dlVP
```

as the formatted string for our example. You must give a string for each piece whose default you want overridden (our example uses the null string for both pieces), and this string must immediately follow either the single or double letter for the piece. You may not have any other letters at brace-level 1 in the format string.

References

[1] Charles Louis Xavier Joseph de la Vallée Poussin. A strong form of the prime number theorem, 19th century.

[2] Donald E. Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1984.

[3] Leslie Lamport. *LATEX: A Document Preparation System*. Addison-Wesley, Reading, Massachusetts, 1986.

[4] William Strunk, Jr. and E. B. White. *The Elements of Style*. Macmillan, New York, third edition, 1979.

[5] Unilogic, Ltd., Pittsburgh. *Scribe Document Production System User Manual*, April 1984. Chapter twelve and appendices E8 through E10 deal with bibliographies.

[6] Mary-Claire van Leunen. *A Handbook for Scholars*. Alfred A. Knopf, New York, 1979.