Productivity Engineering in the UNIX† Environment

An Improved User Environment for TEX

Technical Report

S. L. Graham
Principal Investigator

(415) 642-2059

"The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government."

Contract No. N00039-84-C-0089

August 7, 1984 - August 6, 1986

Arpa Order No. 4871

†UNIX is a trademark of AT&T Bell Laboratories

AN IMPROVED USER ENVIRONMENT FOR TEX*

Peehong Chen *Michael A. Harrison*
Jeffrey W. McCarrell *John Coker* *Steve Procter*

Computer Science Division
University of California
Berkeley, CA 94720, USA

Abstract

This paper describes the enhancements we have made at Berkeley to the TEX environment. The goal of the enhancements is to shorten the edit-compile-debug cycle in preparing TEX documents. An important step in cutting down debugging time is the development of a DVI previewer on a workstation with a high resolution bit-mapped display. Yet another approach we took is the integration of TEX with a powerful display-oriented editor whereby the editing, compiling, and certain pre- or postprocessing of a document may be automated. We present some of the important results of our work in this paper with a general critique on TEX that underscores our motivations.

1. Introduction

This paper is a report on the improvements we have made at Berkeley to the TEX document preparation environment. During the past few years, TEX [8] has evolved at Berkeley as an alternative to the standard UNIX text processing system `troff` [10] and its preprocessors. We enjoy doing our writings in TEX because it has a number of advantages over other systems, some of which we see are its extensibility (macros), mathematics, and the high quality output. Unfortunately, at the same time we have also discovered some disadvantages of and inconveniences in using TEX. The fact that TEX is batch-oriented often makes it very expensive to reprocess a document with only few changes. Another criticism we consider valid is its lack of graphics support, although a "hook" is available (`\special`) and many proposals have been made over the years in the public forum such as the `tex-hax` mailing list.

In 1984, a team was formed at Berkeley to conduct research in document preparation systems, with the improvement of TEX as our primary goal. The work we have done in the project comprises two phases. In phase one we took the obvious approach to make

* This work has been sponsored by the U.S. National Science Foundation under Grant MCS-8311787 and by the U.S. Defense Advanced Research Projects Agency (DoD), ARPA Order No. 4871, monitored by Naval Electronic Systems Command, under Contract No. N00039-84-C-0089. Additional support was provided by the State of California MICRO program under grant number 532422-19900.

enhancements by integrating TEX and its accessory programs with an interactive editor. Furthermore, on our workstations we developed a DVI previewer and other TEX-related tools to shorten the edit-compile-debug cycle. In the second phase, which is still under development, we are taking a more ambitious approach that attempts to design and build a brand new system based on TEX. The idea is to stick with TEX's source language including its macro facility and formatting algorithms but, in addition, making it incremental and more user friendly. Moreover, editing tables, graphics, and raster images will be an integral part of the system. We call this new environment Visually-ORiented TEX, or VORTEX.

The two approaches are actually interrelated. The first phase started earlier with porting TEX to the SUN workstation and developing a DVI previewer under its window system, followed by integrating all TEX-related software in a display-oriented editor. By now it has produced a number of programs which are useful in and of themselves. They have also become important prototypes and special subsystems for the ultimate VORTEX environment.

This paper is concerned with the results produced by phase one of our project, as the objectives and design of the VORTEX system itself is discussed elsewhere [4]. We first give a general critique on TEX from the user's point of view in the next section, pointing out its strengths and weaknesses as compared with other systems. This also serves as a background for later sections which discuss some of the important enhancements we have done.

Section 3 describes the functionalities and technical aspects of `dvitool`, the DVI previewer we have been developing on the SUN workstation. Working with a window-based system, one can have a text editor operating on a source file, also have available a console or shell window in which to run additional jobs, and have a third window displaying the formatted output all at the same time. Our program for previewing DVI files is called `dvitool` which supports keystroke commands, pop-up menus, scroll bars, and other standard user interface in a window paradigm.

Section 4 discusses the TEX integration with GNU Emacs [13]. For the time being, most of us use Emacs as our editor-of-choice in preparing source files. Emacs allows one to customize it by writing programs in a Lisp dialect. This turns out to be an extremely powerful language, and we have constructed very large programs which aid in the use of TEX. In addition to doing obvious things such as matching braces automatically, the user can receive a great deal of assistance in working with bibliographies. It is possible to avoid the use of multiple passes with LATEX/BIBTEX [9,11] and a great many other important facilities can be made available through the use of our system. Discussions in this section are concentrated on high-level abstractions of the design and its basic functionalities.

Finally some concluding remarks are given in Section 5 on our experience with building this improved TEX environment. Notes on what we expect to do in the future, especially with VORTEX, will also be mentioned.

2. A Critique

We have chosen to base the VORTEX system on TEX for a number of reasons which center around TEX's unique advantages. One of these is the concept of a device independent file which gives the same results on different output devices, and the only limitation is the

resolution of the device. We also are committed to getting the highest possible quality from our systems. TEX has outstanding algorithms for dealing with the basic problems of computerized typesetting. In particular, the line breaking algorithm is excellent and the hyphenation algorithm gives impressive results for relatively small table sizes.

TEX's greatest strength is its handling of mathematics. It is in processing mathematics that the advantages of a source-based system such as TEX become very noticeable. In a seminar given at Berkeley, students were asked to typeset a page of complicated mathematics from a textbook. This was not terribly difficult to do using TEX. On the other hand, with Xerox Dandelions available some students attempted to set the same page using the processing facilities available as part of the STAR system [1]. It took much longer, and the results were very disappointing in terms of appearance. This has lead us to believe that typesetting mathematics without a source language is in general a painful task. Even with the notion of *plagiarizing*, (i.e. by copying template formulas from what's available in system's database), which is supported by some WYSIWYG (what-you-see-is-what-you-get) systems like LARA [6], the potential tampering by the user will still put its final quality in question. The output produced by TEX on mathematics exceeds the levels of all but the best hand compositors, and it can be truly said to be an "expert system" in the production of mathematics.

Unfortunately, TEX has weaknesses as well. There is no graphic facility whatsoever. The system is oriented to batch operating systems. TEX has facilities for setting tables, but these are primitive, and the construction of tables in TEX is an enormously difficult and time-consuming chore. The situation is somewhat ameliorated with LATEX [9] which makes producing tables almost as easy as using `tbl` and `troff`.

TEX achieves its flexibility by being a macro-based system. That is, the user writes macros to accomplish what one wishes to do. Such examples of course are the `plain` package and the LATEX macro package. There is a very poor human interface in the macro system, and it requires a high degree of wizardry to use it.

While the source based systems have been impressive in the quality of their output for difficult typesetting jobs like mathematics, the situation is reversed with the WYSIWYG editors. Here excellent human interfaces have been developed. Users find it easy to learn the systems for simple word processing or even for the construction of graphics and the preparation of tables. The software for the Macintosh [7] is especially noteworthy in this regard. On the other hand, these systems cannot do mathematics well and they do not generally produce high quality results comparable to those obtainable from TEX.

3. TEX without Paper: Dvitool

`Dvitool`, a TEX output previewer running on the SUN workstation, is an integral part of the Berkeley TEX environment. Our primary goal for `dvitool` was to provide a means to view the DVI representation of a TEX file without printing it. In our large community, printing takes a long time and is particularly frustrating when debugging a macro. The section describes `dvitool`'s basic functionalities, its user interface, and the future directions.

3.1 Functionalities

One of the first things `dvitool` does when executed is look for a user specific customiza-

tion file. The customization file describes initialization parameters which are mostly window system specific, for example, the placement and size of the window dvitool runs in. After dvitool has started up, the image it presents of the DVI page is 1.45 times the size of an 8.5 by 11 inch sheet of paper. This scale factor means that when dvitool is made as big as the screen allows, the full width of the page and about 60% of its height will be visible. The scale factor is largely historical, but it is also practical. It turns out that at our screen resolution (80 dots per inch) that 1.45 times normal size is close to the lower bound of usability. Any smaller and the fonts would be illegible. Even at 1.45 times normal, dvitool's fonts cannot be called satisfactory.

Once the page is painted, the user can scroll an arbitrary amount either vertically or horizontally. The default action is to scroll 1/3 of the window size. There are also commands to position on any edge of the page, so one keystroke positions the bottom of the page at the bottom of the window. The complete DVI page is read in at one time, so that new views of the same page are instantaneous.

The user can move back and forth across pages as well. Since a new DVI page must be read and painted, there is a short delay, typically 4 seconds in our environment. Pages are cached, however, so that once a page has been viewed, viewing it again is nearly instantaneous. The memory penalty for page caching is about 6K bytes per page, which is not too prohibitive on our workstations with 4 megabytes of memory. The user can limit the number of cached pages and dvitool internally sets the limit whenever it cannot obtain enough memory to cache another page.

"Wildcard" searches have been implemented on any of TEX's ten \count variables. These are not full regular expressions; they just match any field so the user can go to the first page in chapter 4, for example. Commands also exist to view the first and last pages of the file. The movement commands are reminiscent of a text editor.

We've also implemented a global magnification scheme in dvitool. TEX's \magnification macro magnifies the size of individual letters on the page, but keeps \hsize and \vsize in true dimensions so the pages always come out 8.5 by 11 inches. Dvitool's magnification, on the other hand, is global. It simply magnifies the entire page. There are 6 steps available, corresponding to TEX's 6 magsteps. This feature is particularly nice for aging eyes. We implemented discrete steps of magnification rather than a continuous spectrum because new magnifications require new fonts.

Dvitool can also report information about the DVI image, though this ability isn't quite as useful as it sounds. DVI files were designed to be a compact representation of a typeset page. There isn't a lot of extraneous information in them, so there isn't much that dvitool can report. About the most useful feature is that the user can select a character with the mouse and ask what font that character is set in. Even this is of limited usefulness, however, because the user has to correlate the font name in the TEX document which may have gone through arbitrary macro expansion to the system's name of the font that dvitool knows about. For example, TEX users in our environment have to know that TEX uses amitt for italic fonts. LATEX users have to correlate amitt with emphasized text as well.

A companion program for dvitool we've developed is called texdvi. As the name implies, in one step TEX is executed and then the output is previewed using dvitool.

Texdvi is smart enough to start up a new dvitool or to signal a running dvitool to preview the newly formatted output. However, dvitool will not be invoked if the DVI file was not changed. In addition, if there were errors during the TEX job, texdvi asks the user if he still wants to preview the potentially flawed DVI file. The new DVI image displays the text at exactly the point that was displayed earlier. This is very useful for debugging because of automatic repositioning. There are similar mechanisms for working with LATEX and SLiTEX (i.e. latexdvi and slitexdvi). In fact the program is set up in a way that with the formatter replaced by any TEX dialect, say FooTEX, the program footexdvi only has to be a symbolic link to texdvi.

3.2 User Interface

The user interface to dvitool has undergone many changes. Our window environment offers many ways to invoke commands. We finally decided on two: keystrokes and menus. The reason is that inexperienced users of dvitool expect to use the mouse to perform commands in a window environment, while advanced users find the menus cumbersome. We provide both so that dvitool is both easy to learn for the novice and responsive to the expert. We provide clues to help the user graduate from novice to expert level. For example, all of the menu commands also contain the matching keystroke commands as a hint to the user. We also provide an on-line help facility which is itself a DVI file.

We rejected having "buttons" as our user interface. Buttons can be thought of as menus which are statically displayed. They are fixed areas inside the window that the user points to and clicks on with the mouse to invoke a command. The standard placement for buttons is a row of them either across the top or down one side of the window. The idea was rejected for two reasons: we wanted to devote as much screen real estate as possible to displaying the DVI page, and we didn't want to force the user to be continually switching from the keyboard to the mouse.

As part of dvitool's customization facility, keystroke commands can be redefined by the user to look like the key bindings of his/her favorite text editor. This feature is particularly important because users frequently switch from the editor to dvitool and back.

3.3 Future Directions

Dvitool was developed on the SUN workstation and runs under their proprietary window system [2]. Some care has been taken to isolate the system dependent parts of the code, but any program which must deal intimately with a non-standardized graphics interface is inherently not very portable. Dvitool is typical in this respect. We expect to begin work on a port to the X window system [5] soon.

Over time, the the user interface to dvitool has become more editor-like. Since it is possible, and indeed desirable, to have both your text editor and dvitool on the screen at the same time, we've tried to make them as homogeneous as possible. Planned additions to dvitool include negative magnification (shrinking) and a word search facility. Ligatures present problems for the word search routines. At the DVI level, ligatures such as the two characters "ff" are printed as a single character. Certainly we could create a translation table at compile time to do that mapping, but that solution is necessarily dependent on external and potentially changeable information.

Another problem is how to search for math text. How would the user tell `dvitool` to look for x_3, for example? The obvious solution of having `dvitool` recognize the TEX syntax for that expression implies that `dvitool` would have to be able to parse the TEX language which is a task far beyond its scope.

4. Integrating TEX with Emacs

One way to enhance the TEX environment is to customize a display-oriented text editor whereby the editing, compiling, and certain preprocessing or postprocessing of a document may be automated. Since in general a modern display editor is interactive, this approach turns out to be a remedy for TEX's lack of interaction with the user. Our editor of choice is GNU Emacs [13] which is the latest implementation in the Emacs family of editors [12]. GNU Emacs supports Emacs Lisp (or ELisp) in both interpreted and compiled forms. ELisp is very close to a full Lisp implementation: general list and attribute processing are available as part of some 900 system primitives and functions for various editing purposes.

The enhancements we've made to TEX in Emacs are basically two macro packages: ***TEX-mode*** and ***BIBTEX-mode*** [3]. The combined system is about 6,000 lines of ELisp code which is split into eight different files according to functionalities. Only the most essential parts are loaded initially; other files are loaded on demand. The first package, *TEX-mode*, is an aid to editing, spelling checking, compiling, previewing, and printing TEX and LATEX/SLITEX [9] documents. *BIBTEX-mode*, on the other hand, is an interface to editing BIBTEX [9,11] databases. Perhaps more importantly, the two modes are integrated to yield a very nice bibliography system for both types of documents.

A major focus of our design is a clean and uniform abstraction for both document structure and desired functionalities. The document structure refers to the types of objects and their interrelationships in a document that must be made explicit to the user. Functionalities are the possible operations which may be performed on certain objects. The two are bridged together by a set of commands which is uniform across the board in terms of naming and key bindings. Because there are so many commands in the system, the hope is to make them not only useful but easy to remember as well.

4.1 Document Structure

At the source level, *TEX-mode* makes the distinction between a *document* and a *file* by acknowledging that a TEX or LATEX document may involve multiple files connected by `\input` or `\include` commands. *TEX-mode* views a document as a tree of files with edges being the connecting commands. The root of a document tree is called the *master file*. Operations involving the entire document must be started from the master file. The processing sequence is the preorder traversal of the tree. In *TEX-mode*, each individual file has a link to the master to assure any global commands initiated in its buffer will always start from the master. The link to master also makes it possible to separately compile any component file or a part of it. The technique used in *TEX-mode* to do separate compilation is discussed in Section 4.2.4.

The next level of abstraction is a *file*, or when loaded in Emacs, a *buffer*. Objects of even smaller granularities include *regions* and *words*. A *region* is a piece of text, including any white space, bounded by a marker and the current cursor position (i.e. *point* in GNU Emacs). A *word* in *TEX-mode* is a piece of text with no white space in it.

At the output DVI level, the distinction is less complex. The only abstractions are the DVI file as a whole and subranges of one extracted out as another file. Normally DVI files themselves are not visited in Emacs. Therefore in a buffer bound to the TEX source `foo.tex`, the implicit operand for operations such as *preview* and *print* is `foo.dvi` instead of `foo.tex`. With the abstractions, it is possible to preview or print a DVI file partially as well as in its entirety.

Furthermore, *TEX-mode* maintains the notion of *document type* which may be either TEX, LATEX, or SLITEX in our current version. The type information is needed when the user tries to execute operations involving programs which are type-specific, such as the formatter (i.e. `tex`, `latex`, or `slitex`) and the document filter (i.e. `detex` or `delatex`). However, such information is implicit to the user except for the first time — once specified it will be saved as a comment line in the document to be read by later invocations. In other words, from the user's point of view, operations in *TEX-mode* are generic. For instance, an operation is known as *format* at all times instead of as `tex`, `latex`, or `slitex` under different situations. *TEX-mode* does operator overloading implicitly by consulting the type information.

4.2 Functionalities

Operations in our enhanced TEX environment fall into one the following categories: (1) delimiter matching, (2) bibliography processing, (3) spelling checking, and (4) compiling-debugging-previewing-printing. All four are defined in *TEX-mode* with the exception that the second also relies on *BIBTEX-mode*.

4.2.1 Delimiter Matching

A rather complete delimiter matching mechanism is implemented in *TEX-mode*. First, automatic delimiter matching applies to a pair of parentheses, brackets, or braces (i.e. `(...)`, `[...]`, or `{...}`). That is, whenever a self-inserting closing delimiter (i.e. `)`, `]`, or `}`) is typed, the cursor moves momentarily to the location of the matching opening delimiter (i.e. `(`, `[`, or `{`). *TEX-mode* gets this for free simply by modifying Emacs' syntax table entries.

The matching of other delimiters is less straightforward. Matching delimiters such as quotes (i.e. `'...'`, `''...''`, and `"..."`) and TEX dollar signs (`$...$`) cannot be done automatically by syntax entry modifications. For example, the symbol `'` is the right quote as well as the apostrophe. Modifying syntax entries in the normal way is inappropriate because we don't want the cursor to bounce in the case of apostrophes. Matching double quotes (`"..."`) and TEX dollar signs (`$...$`) is even harder because the opening and closing delimiters are identical in those situations.

Semi-automatic Delimiters

TEX-mode introduces the notion of *semi-automatic matching*. To get semi-automatic delimiters inserted, one types some special commands and the text between a bound and the current cursor position will be enclosed by a pair of delimiters. The bound may be explicit or implicit. For the first case, which is called *zone matching* in *TEX-mode*, the user consciously sets a zone marker and closes it at the other end by typing the command that corresponds to the delimiters wanted. For the second case, an implicit bound refers

to the white space before or after a word. *TEX-mode* calls this scheme *word matching* . Symbols like `'...'`, `''...''`, `$...$`, and `$$...$$` as well as groupings for fonts and boxes such as `{\it ...\/}`, `{\tt ...}`, `\hbox{...}`, and `\vbox{...}` are all built-in semi-automatic delimiters in *TEX-mode*. For instance, typing the command C-c i (`tex-word-it`) will automatically enclose the previous word in `{\it ...\/}`, with ... being the word.

Automatic Delimiters

Matching identical opening and closing delimiters is a difficult task. The situation is further complicated by the TEX dollar sign because a pair of single dollar signs (`$...$`) denotes *math mode* in TEX whereas a pair of double dollar signs (`$$...$$`) means *display math mode*. A correct mechanism not only has to know which self-inserting `$` or `$$` is an opening delimiter and which is a closing one but also must be clever enough so that the second `$` in `$$` does not match the one preceding it. Furthermore, a dollar sign may be escaped (i.e. `\$`) in TEX which must be treated as an ordinary symbol rather than a math mode delimiter. *TEX-mode*'s dollar sign matching mechanism is designed to handle all cases correctly. A similar but less complex mechanism applies to the matching of double quotes (`"..."`).

LATEX Delimiters

One of the most commonly used commands in LATEX is a pair of `\begin` and `\end` which is normally used to embrace a large piece of text under a certain *environment*. Environments can be nested in the obvious way, just as in any block-structured language. With several levels of environments in place, proper indentations become essential to readability.

TEX-mode has a facility that opens and closes LATEX environments automatically with proper indentations inserted. For example, with one command, you can get

```
\begin{enumerate}
  ▮
\end{enumerate}
```

where ▮ denotes the current cursor position. Most LATEX environments are predefined and there is an operator which prompts you for an environment and its associated arguments interactively.

Customization

All delimiter matching schemes mentioned above can be customized. A new pair of delimiters may be defined statically by putting the information in Emacs' profile (`.emacs`) to have it available for every *TEX-mode* session. Alternatively, the user can enter the information interactively to *TEX-mode* so that a new delimiter pair is bound for only one particular Emacs session.

4.2.2 Bibliography Processing

BIBTEX is a bibliography preprocessor for LATEX documents. Under the LATEX paradigm, one makes citations in the source by referring to entries defined in bibliography databases.

A BIBTEX database is a file with the name suffix '.bib' which contains one or more bibliography entries. To get the final output, the user first runs `latex` on the source to produce some reference information which is passed to `bibtex` to generate the actual bibliography. A second run of `latex` looks up the bibliography and produces cross reference information based upon which the last `latex` does the actual substitutions.

TEX-mode makes it possible for BIBTEX to work on plain TEX documents as well. It bypasses the first `latex` and invokes `bibtex` directly. It works with multiple files involved in a document by recursively examining `\input` commands in plain TEX files and `\includeonly`, `\include`, and `\input` commands in LATEX files. Furthermore, it prompts you for corrections at the places where citation errors are found. In addition, a database lookup facility is available for making citations. The implications are:

1. The same mechanism not only works for LATEX documents which BIBTEX was originally designed for but for plain TEX documents as well.
2. To get the final LATEX output, the user only has to invoke one or two `latex`'s manually — *depending on non-citation symbolic references being present. To get the final* TEX output, only a single run of `tex` will suffice. This is due to the automatic invocation of `bibtex`, the error correcting facility, and the automatic substitution mechanism.
3. Due to the lookup facility, the user does not have to memorize or type in the exact entry names in order to make citations. The system prompts you one by one the matching entries found in the specified bibliography database. The selected entry will be interpolated into the source automatically.

In our environment there is a powerful IBM 3081 mainframe which we often use for large jobs. Currently it supports TEX and LATEX but not BIBTEX. However, using our bibliography system in Emacs, we are able to get all bibliographical references resolved in our local machine (VAX/UNIX), send the document as a single file to the 3081 computer over the network, execute `tex` or `latex` there, and get the resulting DVI file back. In fact, a fair amount of work needs to be done before a `.tex` file is sent to the remote machine. For instance, the conversions between special characters used in the two systems and between stream-based files (UNIX) and record-based files (IBM 3081 takes 80 columns per line) must all be realized beforehand. It would be impossible to take advantage of the remote machine's fast speed if our preprocessing facility were not available.

Finally, bibliography database files can be manipulated using *BIBTEX-mode*. It has all fourteen BIBTEX bibliography entry types predefined so that to insert a new entry the user only has to specify its type. A skeleton instance of the specified type will be generated automatically with the various fields left empty for the user to fill in. A set of supporting functions such as *scroll*, *field copy*, *entry duplicate*, ..., etc. is provided to facilitate this content-filling process. Other major features of the mode include a facility to make a draft bibliography for debugging and previewing purposes and an extended abbreviation mechanism that allows one to abbreviate chunks of text and to browse abbreviations defined in any `.bib` files.

4.2.3 Spelling Checking

The *TEX-mode* spelling interface allows one to check the spelling for a word, a region, a buffer, or the entire document. It is specially tailored to TEX and LATEX documents:

keywords and commands of TEX will first be filtered out by a program called detex and those of LATEX by delatex, before being sent to the system's spelling program. The user will be able to scroll the list of misspelled words and make corrections, including using a dictionary lookup facility. To avoid screen redisplay overhead, searching under low speed connections ($\leq$ 2400 baud) is implemented for scrolling and replacing any misspelled words in the buffer. That is, if an instance is not visible in the current window, only the line containing it is shown in a tiny window at the bottom of the screen.

4.2.4 Compile-Debug-Preview-Print

A number of TEX related programs can be invoked from *TEX-mode*. These external programs are executed uniformly in Emacs' inferior shell process. The generic operators are *format*, *display*, *view*, and *print*. The first two are overloaded based on the document type. The *display* operator is a pipeline of formatting followed by previewing (i.e. texdvi, latexdvi, or slitexdvi). The *view* operator is bound to a previewer such as dvitool described in the Section 3. The other two operators are self-explanatory.

The notion of master file plays an important role here. Both *format* and *display* operate on either the entire document, a buffer, or a region in buffer. A document preamble and similarly a postamble can be associated with the master to contain the document's global context. To separately compile a component file or its subregion, a mechanism is available in *TEX-mode* that includes in a temporary file the document's preamble and postamble with the selected text inserted in between. The system will then run *format* or *display* on this temporary file. This technique is primarily for debugging purposes as there is no provision for linking separately generated DVI files into one big DVI file. However, for users wanting only a quick look at a relatively small portion of a document in the debugging phase this automatic facility turns out to be very valuable.

A DVI file can be previewed or printed in its entirety. *TEX-mode* can also invoke the program dviselect so that arbitrary pages within a DVI file may be extracted and only these selected pages will be previewed or printed. This is another useful tool for avoiding unnecessary work in a batch oriented environment like TEX.

Another support for debugging is a mechanism which automatically positions the cursor to the line and column where the error occurs. Also available are little tricks like commenting out a region by a single command and recovering it by another command.

4.3 Commands

The bridge between objects and their corresponding operations is the set of commands available to the user. The central issue here is the uniformity in both naming and key bindings.

By and large, the two modes obey the naming convention that a function name consists of three parts: prefix (tex- or bibtex-), generic operator, and abstract object. The corresponding key binding will be the C-c prefix, followed by C- and the first letter of the middle part, then the first letter of the last part. One example is the *TEX-mode* function tex-format-document with its corresponding key binding being C-c C-f d. *BIBTEX-mode* deals with a simpler set of objects such as bibliography entries and their component fields.

[7] L. Johnson. *Macintosh MacWrite Manual.* Apple Computer, Inc., Cupertino, California, 1983.

[8] Donald E. Knuth. *The TEX Book.* Addison-Wesley Publishing Co., 1984.

[9] Leslie Lamport. *LaTeX: A Document Preparation System. User's Guide and Reference Manual.* Addison-Wesley Publishing Co., 1986.

[10] Joseph F. Ossanna. *Nroff/Troff User's Manual.* Computer Science Technical Report 54, AT&T Bell Laboratories, Murray Hill, New Jersey, October 1976.

[11] Oren Patashnik. *BIBTEXing.* Computer Science Department, Stanford University, Stanford, California, March 1985.

[12] Richard M. Stallman. *EMACS: the extensible, customizable self-documenting display* editor. In *Proceedings of ACM SIGPLAN/SIGOA Symposium on Text Manipulation*, pages 147–156, Portland, Oregan, June 8–10 1981.

[13] Richard M. Stallman. *GNU Emacs Manual.* Free Software Foudation, Cambridge, Massachusetts, second edition, March 1986.