

Why I Use the L^AT_EX Typesetting System*

Michael C. Wendl[†]

School of Medicine, Washington University, St. Louis MO 63108, USA

August 16, 2004

Abstract

I discuss my rationale for using a typesetting system, in this case L^AT_EX, as opposed to what are commonly called “word-processors”, e.g. MICROSOFT WORD[®]. Although arguments are made largely from a technical standpoint, the explanatory material should be enough to guide even the casual computer user through. Hopefully, you will find this material useful in better determining your own preferences for handling textual content.

*As the title would imply, this essay was written using L^AT_EX. Thanks to David Dooling of Washington University School of Medicine for feedback. Permission is granted to copy, distribute and/or display this document under the terms of the Attribution–NoDerivs–NonCommercial License, Version 1.0 or any later version published by Creative Commons. For more information, contact Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA. You are reading version 1.01 of this document.

[†]e-mail: mwendl@wustl.edu

1 Introduction

Computers have fundamentally changed the ways in which we perform many tasks, especially how we prepare and present textual material. Such material has traditionally appeared on the printed page, but now can also be presented electronically, e.g. as a web page, an e-mail, or as an electronic emulation of a paper document viewed on a computer display.

The task of creating all such documents is divided into 2 logical parts:

1. composing the content itself, i.e. formulating the sentences, equations, figures, tables, etc. that comprise the document, and
2. typesetting the material such that the final product presents itself to the reader in the desired format.

In the years BC¹ these steps were observed out of necessity. An author wrote a manuscript, into which “mark-ups” would be inserted to direct the typesetter how the final document should appear in print. Mark Twain did not particularly worry about font size and margins — only about content. Nowadays, computers give us the capability of acting as both author and typesetter. Indeed, we would probably not want to return to the slow manual process of yesteryear.

There are many avenues for handling the task of document generation on the computer. Here, I will try to point out some of the advantages of true typesetting, as opposed to what is commonly called word-processing. One distinction is that the former observes the above steps, while the latter combines them. In my own case, I had originally opted for using word-processors, one could say, by default. We had an early version of MICROSOFT WORD[®] in the laboratory where I was a graduate student. I was basically unaware of any alternatives short of my mechanical typewriter, to which MICROSOFT WORD[®] was an unquestionable improvement. Over a period of time, I transitioned through various systems, finally arriving at L^AT_EX, which I have used for a number of years. I continue to have the distinct feeling of “how could I have ever lived without this?”. However, this attitude only reflects better features, which I will outline below, that the user realizes.

There are also other significant, but less obvious advantages related to storage, longevity, security, and control over one’s content that make a typesetting program more appropriate for all of

¹Before Computers

my personal needs. *I realize that your needs may be different.* My views are shaped by the work I do as a scientist and the associated capabilities necessary to present scientific work, correspond with colleagues, etc. However, my observations over a number of years also suggest to me that most people, for the type of documents they deal with, would be better off using a typesetting system rather than a word-processor. If your activities are limited to “one-off” documents, e.g. fliers or party invitations, a word-processor, or better yet an application specifically designed for these types of documents, would clearly be the better choice. This is not meant flippantly. *Ad hoc* documents are usually very short, focus on raw formatting, and have essentially no logical structure. Moreover, their “lifespan” is short — one isn’t typically concerned about being able to read a party invitation on one’s computer, say 5 years after the fact. However, these issues can indeed be important for other types of textual content. Let’s start with brief descriptions of the relevant software entities we’ll be speaking of.

1.1 Text Editors

A text editor is a computer program that manages the generation, editing, and storage of raw textual content (step 1). Little or no facility exists for structured typesetting (step 2). There are numerous, perhaps hundreds of such programs, for example `vi` (standard on Unix[®]-type operating systems), `pico` (also on many Unix[®] installations), the popular GNU Emacs editor², as well as many commercial implementations, like MICROSOFT NOTEPAD[®]. This is probably a good place to introduce the concept of standards as they relate to electronic information. Plain text is represented electronically according to the *American Standard Code for Information Interchange*, universally abbreviated³ as ASCII. An important aspect of text editors is that they *all* handle ASCII, without exception. Content is thus independent of the program used to create it. A message composed in

²Available from the GNU website <http://www.gnu.org>.

³The abbreviation “ASCII” is pronounced *as’-kē* and describes a standard code for how computers represent the English character set numerically. Such representation is necessary because computers ultimately understand only numbers. For example, upper-case “M” corresponds to 77. The term ASCII is used generically as a synonym for textual content. There are actually a number of more generalized standards, e.g. the ISO-8859-1 Latin character set. For our purposes, we shall simply observe the convention of referring generically to all text as “ASCII” or “plain-text”.

any text editor can be read by anyone with any other text editor. It can be read by all e-mail clients, printed on all printers, etc. Plain-text is the universal format.

1.2 Word Processors

Here, we use the term “word-processor” to refer to WYSIWYG software⁴. These packages, of which MICROSOFT WORD[®] is clearly the most successful commercial example, combine the tasks of content generation (step 1) and typesetting (step 2). As you type, your text is rendered in real-time in its processed form. In this context, mechanical typewriters are also word-processors because you format as you type, e.g. for the right margin, section headings, etc. In commercial variants, the information is saved on your local hard disk in a proprietary binary format. The resulting content can be accessed only through the specific word-processor that created the original document⁵. A person without the word-processor used to create the document may be unable to read it.

1.3 Typesetting Software

A typesetting program, e.g. L^AT_EX, is one which takes plain-text content as *input* (step 1), processes, i.e. typesets it (step 2), and then writes a formatted file as output. One or more of the input files will contain the raw textual content, along with textual mark-up commands that direct the program to perform the desired formatting. In the simplest case, e.g. a personal letter, there will be a single input file. For large-scale projects like a reference textbook, there could be many input files, for example each book chapter might reside in its own file. All of the input files are created with a text editor, not the typesetting program itself. Output can be in a variety of viewable forms.

⁴What You See Is What You Get — this is a common acronym and is pronounced *wiz'-ē-wig*. It alludes to the concept that one sees their formatted content in real-time.

⁵In actuality, access may depend further on having a compatible version of the processing software. For example, a person using an older version of a word-processor may not be able to read an e-mailed document that was created using a more recent version. Portability can also depend upon similarity of computer hardware. Newer open-source word-processors, for example OPENOFFICE (<http://www.openoffice.org>), largely avoid this problem by storing content in a standard textual form. The issue of portability is discussed in more detail in §2.3.

1.4 Printers and Printer Languages

The end-goal is obviously to present your formatted textual content, including equations, tables, and figures, to your audience. This often takes the form of printed output. Printers do not actually understand the various processed formats we have alluded to above. They have their own *page description* languages that handle the actual printing process⁶. Consequently, printing a document necessarily includes a conversion step⁷ from the native format, e.g. the binary word-processor document format, to a printer language.

1.5 Other Peripheral Software Entities

We have now identified the main software players in this scenario. Equally important, we also now see that there are a variety of associated file formats and languages. This implies there should be programs that convert from one format to another, which is in fact the case. We do not discuss these programs, except to point out that they provide some level of inter-operability among the ways of presenting content. For example, any raw L^AT_EX document can be automatically converted into a web-based document using a free program called `latex2html`.

We will also assume that you have a favorite program(s) for creating, editing, and managing any graphics that go into your documents, for example photographs, line drawings, x-y plots, etc. There are numerous such programs, whose merits we do not have space to discuss here. However, it is probably safe to assume that your tool supports some of the standard graphics formats, so that any output can be imported into your document.

⁶ There are a number of such languages, of which probably the most widely used today is the Portable Document Format (PDF). An important aspect of page description languages is that the representation, and thus the output of a particular document is independent of the original hardware, software, and operating system used to create it. Because PDF is readable by lots of different viewing software and is printable on all printers, it can be considered “universal” in the sense that there are no current limitations in accessing PDF documents. It is not truly universal in the same sense as plain-text because it could eventually be displaced by a better language. In the absence of conversion to the newer language, old documents would no longer be widely accessible.

⁷This process usually happens “under the hood” without the user typically being aware of it.

2 Comparison of Approaches

Now we get to the substance of this discussion. Here, I will describe in some detail the issues I consider to be the most important in determining what approach to use for managing textual content. We will draw comparisons among several specific software packages. Again, I will give the disclaimer that these are my opinions and I appreciate that your needs may not be the same. My personal document-generating activities fall largely into the following categories:

- Correspondence — I write lots of letters to colleagues, editors, etc.
- Scientific Papers — My work is submitted to scientific journals in the form of manuscripts, which invariably contain figures, tables, and some amount of mathematics
- Presentations — I also present this same work in seminars and classes in the form of overheads
- Texts — I write book-length notes for each course I teach and other lengthy reports
- CV — Like almost everyone, I maintain an up-to-date résumé
- Tests — in my classes, I give tests, which have their own customized formatting

In no particular order, the issues are as follows:

2.1 Versatility

All the document types I just listed are similar only in the sense that they are ultimately designed for print media, e.g. paper or overhead transparencies, or the electronic emulation of a paper display. Otherwise, their structure and formatting are all different. You might suspect that different computer applications would be needed to handle these, and to some degree this is, in fact, the convention. For example, folks will frequently use MICROSOFT WORD[®] for correspondence, short papers, and CVs; MICROSOFT POWERPOINT[®] for presentations; and ADOBE FRAMEMAKER[®] for large structured documents, like reports and books⁸. Each of these three applications handles

⁸ ADOBE FRAMEMAKER[®] is apparently being phased-out by its manufacturer. In fact, it was discontinued altogether for the Apple platform on April 21, 2004. Most observers speculate it will gradually be discontinued on other platforms as well. See §2.4 for a discussion on the issue of software longevity.

a specific document type, which the others are not good at dealing with. Unfortunately, this requires the user to master a number of different software applications in order to accommodate their text-processing needs.

Conversely, L^AT_EX is explicitly designed to handle all of these document types equally well. Users need only to know *one* tool, rather than many. In part, this functionality is due to L^AT_EX's concept of pre-defined document classes. However, there are quite a few other native characteristics that enable its versatility. For example, an index can automatically be created based on tags in the input. Likewise, a table of contents can be automatically created based on section headings. These features are obviously desirable for books and long reports. One can also take advantage of automatic numbering and *referencing* for equations, figures, tables, footnotes, etc. and automatic creation of citations and their corresponding bibliography. Fancy presentation features can be implemented for overheads, like incremental bulleting, hot links, and various types of slide transitions⁹.

Although a few of these features can be realized with word-processors via add-on software packages (see §2.6), there are yet deeper levels of functionality that extend L^AT_EX's versatility even further. Suppose we want to simultaneously manage different versions of the same document. For example, it is fairly common to maintain one's résumé in a number of parallel forms, e.g. a concise version, industry-specific versions, and the full version having all the minute details of your career history. Similarly, you might want to maintain two versions of an exam, one for the students which has the questions, and the other for posting which has the solutions (questions plus answers). Version control for such documents quickly gets out of hand if using a word processor. Each version must be stored as a standalone document. When changes are required, each document must be systematically examined and the appropriate edits made. If using L^AT_EX, one simply stores all the information in the same document file, along with simple commands that allow one to choose which version should be created at compile time.

2.2 Typesetting Capability

Generally, the superiority of L^AT_EX's typesetting capabilities as compared to those of word-processing packages are widely recognized. This is underscored by a brief, but compelling observation: com-

⁹L^AT_EX's `prospcr` class is especially recommended here.

mercial publishers often print journals and books directly from L^AT_EX output¹⁰. There are *no* commercially-prepared publications, at least none that I am aware of, that are printed from word-processor output.

L^AT_EX's underlying typesetting engine¹¹ was originally designed to handle what is surely the most challenging aspect of textual presentation: mathematics. The complexity of mathematical expressions that can be typeset is rather remarkable¹² and the results are considered superior from a visual standpoint as compared to those implemented with a word processor. The latter point is *not* my subjective opinion — L^AT_EX arguably has its most ardent following in the mathematics research and publishing community, largely because of this capability. There are quite a few other aspects that also differentiate the visual quality of typeset documents. For example, L^AT_EX has excellent implementations of the more subtle typesetting refinements, such as ligatures¹³, kerning¹⁴, and word placement and justification. Its ability to automatically place floating objects, such as figures, is quite good, but it also allows complete manual control of these aspects, as well. The same is true of hyphenation. Typesetting complicated tables, footnotes, etc. is also rather straightforward.

2.3 Longevity and Portability of Documents

A lot of written work takes the form of “long-term” documents. In other words, a given document has a long life span: it may be read or revised over a period of *years*. This implies that one needs to be able to access documents from a long time ago. Although most people do not explicitly realize

¹⁰This is seen most often in the realm of technical and scientific textbooks. See for example Toro (1999).

¹¹This system, called T_EX, was originally developed by Donald Knuth of Stanford University and is considered complete in the sense that no new features have been deemed necessary since development was “frozen” around 1985.

¹²Most L^AT_EX-related documentation and books show how to typeset a variety of complex mathematical equations and notations. See e.g. Kopka and Daly (1999).

¹³Ligatures are spatial unifications of (usually) two characters that have common features. For example, the two characters **fi** can be represented individually, as in “fi”, or as a ligature, as in “fi”. Professional publishers consider the latter form more pleasing to the eye.

¹⁴Like ligatures, kerning is a tool used to improve the visual presentation of text. Here, spacing is adjusted for certain combinations of letters that would otherwise appear too far apart if left alone. For example, the two characters **WA** can be represented plainly as “WA”, or with modified spacing derived from kerning, as “WA”. Professional publishers consider the latter form more pleasing to the eye.

it, many of the documents they write will fall into this category. For instance, they might need to read old correspondence or continually update a technical document. With L^AT_EX, longevity and portability pose no special issues because one can always go back to the plain-text source file. Recall that plain-text is universal, so we can use any text editor on any operating system running on any hardware platform to access the raw document. These aspects *can* pose problems for word processors.

Consider document longevity. Word-processors store documents in a proprietary form¹⁵. Suppose you want to access an old document, but you have since upgraded your system¹⁶, as well as the actual word-processing program used to originally create the document. *There is no guarantee that you will be able to correctly read your old document.* In the worst case, you would not even be able to open the document. In more typical situations, the document will open, but it displays and prints differently from the original¹⁷. The public at-large is becoming more cognizant of the problem (Baig, 2004). For example, it has been pointed out that much of the technical and scientific information distributed informally via the web in word-processor formats will not be readable 10 or 20 years from now (Wilke, 2004).

This behavior is a consequence of the fact that software makers periodically change their proprietary file formats. The more cynical among us contend that these ongoing modifications are designed to force the consumer into constantly buying software upgrades. However, one also must remember that these companies are under some amount of competitive pressure to make their products ever-more appealing to the consumer with respect to new features, capabilities, etc. These modifications will not always be compatible with older features, thus necessitating changes. The difficulty is not so much the changes themselves, but rather the fact that documentation of these changes is not made available to the general community. In fact, descriptions of these changes are typically considered corporate secrets. Here lies the real difficulty with “proprietaryness”: What

¹⁵As mentioned in footnote 5 (pp. 4), many of the open-source word-processors, for example OPENOFFICE (<http://www.openoffice.org>), avoid this problem by storing their content in a standard textual form.

¹⁶Often this will mean newer hardware from a different manufacturer and an operating system that is upgraded from the previous one, or perhaps different altogether.

¹⁷ See, for example, the essay “Long Term Storage of Electronic Data and Documents” by Thomas Schneider (<http://www.fred.net/tds/longrange.html>). Mathematical expressions seem to be especially vulnerable.

are widely perceived to be standards, e.g. the MICROSOFT WORD[®] .doc format, are, in fact, not standards at all. They are subject to change in unknown, and more importantly undocumented ways. Users must keep buying upgrades to avoid the associated problems.

One is liable to have similar problems with the portability of documents. That is, a document created on one particular system (hardware, operating system, and word-processing software) may not be easy to access on a different system. This issue can arise in a variety of cases, for example when users e-mail documents to one another, distribute documents over the World Wide Web, etc¹⁸.

The printed output itself can also change unexpectedly, even when the sender and receiver are using the same version of the word-processor, because the software may silently reformat the document based on the receiver's local printer settings. In other words, these documents are *not* device-independent in the same way PDF documents are¹⁹. This would pose particular problems for form-type documents that depend upon very specific placement of elements on the page.

2.4 Longevity of Software

This aspect is closely-related, but not identical to that discussed in §2.3. In that section, we pointed out that there is a pressure on the consumer to keep abreast of software upgrades in the interest of document longevity. Here, we are concerned about the longevity of the software itself. The possibility of a program being discontinued by a software maker is usually not considered by the average user, but it does happen.

We raised the example in footnote 8 (pp. 6) that ADOBE FRAMEMAKER[®], a very popular publishing program, has been discontinued for the Apple platform. This has created quite a bit of

¹⁸In my opinion, the issue of portability does not pose as great a problem as that for document longevity. Those who receive documents often do have compatible systems with those who created them. For example, even if the recipient does not have MICROSOFT WORD[®] installed, open-source (and free) word-processors can often handle native MICROSOFT WORD[®] documents acceptably well. Again, as Tom Schneider points out (see footnote 17, pp. 9), mathematical expressions will unfortunately be more problematic. The portability problem can also be addressed, albeit in a slightly less-elegant fashion, by simply converting the document to (universal) plain-text or PDF format (see footnote 6, pp. 5) before it is distributed.

¹⁹See footnote 6 (pp. 5).

distress for users who must now scurry for a new product and convert all their existing documents²⁰. One might presume that the final version of the program could simply be used *in perpetuum*, but this idea fails to consider the dependencies on operating systems, etc., which continue to change. For example, ADOBE FRAMEMAKER[®] 7.0 is the last version that will run on the Apple platform, but it is *not* natively compatible with the current Apple Mac OS-X[®], operating system. The more recent version, ADOBE FRAMEMAKER[®] 7.1, does not run at all on Apple computers. Apple users are out of luck here.

In theory, such a fate can befall *any* software package. However, for popular open-source projects, such as L^AT_EX, this is highly unlikely because of the large base of users *and* freely-associating developers and maintainers. Commercial software, by contrast and by definition, does not have the latter.

2.5 Document Size

The physical size of document files is another problematic issue for commercial word-processing software. Proprietary file formats, although implemented as binary storage, have built-in overhead that dramatically increases the size of document files. Some might object as to whether this really remains an issue in the modern age of enormous disk drives and growing broadband service. However, it is still very easy to find cases where “size matters”, for example the user who must download documents through a slow dial-up connection.

In terms of a specific example, I analyzed a simple four-page publishing contract I recently received from a commercial publisher. The contract document was created with MICROSOFT WORD[®] and e-mailed to me by the publisher. It was all text, except for two small black-and-white corporate logos at the top. The size of this file was 1,384,960 bytes. The size of the actual PDF file that was printed on my local printer²¹ was a mere 39,398 bytes — less than 3% of the word-processor file size. To put this more into perspective, the MICROSOFT WORD[®] file was 3,415% larger than what was required to properly represent the printed pages of output!

²⁰See, for example the discussion thread

<http://apple.slashdot.org/article.pl?sid=04/03/24/1512211&mode=thread&tid=179&tid=185&tid=190>

²¹Recall from footnote 6 (pp. 5) that PDF is the language that your printer understands.

It gets worse. The actual text in the file comprised a mere 15,100 bytes. If we conservatively estimate the logo sizes as 4,000 bytes each and conservatively allow for an extra 2,000 bytes of L^AT_EX formatting commands within the file, we could have implemented this entire document in L^AT_EX with only about 25,000 bytes. The MICROSOFT WORD[®] document would now be 5,440% larger than the corresponding L^AT_EX document. Of course, differences will not always be this dramatic. This is just an illustration of how bad things can actually be.

2.6 Monetary Cost

This is a very simple and straightforward issue. Commercial word-processors are expensive and L^AT_EX is free.

Actually, this may be simplifying things a bit too much. To get a more specific idea of costs, let us step back and look at the so-called Total Cost of Ownership (TCO). Here, we will again compare MICROSOFT WORD[®] to L^AT_EX. With MICROSOFT WORD[®], we have to purchase the actual program, as well as other “add-ons” to obtain necessary capabilities that the program itself does not supply. For simplicity, let’s assume that the only add-on we require is the commercial ENDNOTE[®] package²² for automatically managing bibliographies²³. Also, we will assume that we use the software over a long period of time, such that we eventually will also purchase a software update from the vendor²⁴.

Now let’s talk specifics. To get MICROSOFT WORD[®], we actually have to buy the whole suite of office software. In my own case, were I not using L^AT_EX, I would have to opt for the professional version of MICROSOFT OFFICE XP[®]. This contains the word-processing program, as well as other “office” tools that one could otherwise obtain for free, e.g. for spreadsheets, e-mail, presentations, and databases. The package lists for \$499.99, but I found it for \$279.99 without much difficulty on a discount web site. The ENDNOTE[®] package lists for \$299.95. Now, let us assume that we would eventually upgrade from MICROSOFT OFFICE XP[®] to MICROSOFT OFFICE 2003[®] (again,

²² From Thomson ISI ResearchSoft, an operating division of Thomson ISI[®].

²³ Recall that this is a native capability in L^AT_EX.

²⁴ In fact, this has recently happened with the launch of MICROSOFT OFFICE 2003[®], which supersedes MICROSOFT OFFICE XP[®].

the professional edition). I found this upgrade on another discount web site for \$249.99.

In this scenario, we would be spending a total of \$830 to outfit our computer with a word-processor and some basic add-on functionality. The total cost using L^AT_EX: \$0.00. L^AT_EX is truly free in the monetary sense²⁵. When replacing your existing computer system with a new one, you can very likely expect to have to re-purchase all these programs over again.

2.7 Security

By “security”, we are talking specifically about eliminating, or at least minimizing the ability of computer viruses to damage your filesystem, to disseminate themselves, and to carry out other malicious activities. There is *no* inherent security risk in using L^AT_EX because the files one works with are simply plain-text files. They have no facility to unwittingly harbor a virus.

Conversely, there is quite a conspicuous lack of security associated with many word-processors. This stems from their usage of “programming macros”, which were originally designed for the legitimate purpose of automating repetitive tasks for the user. However, macro programming languages have access to certain operating system functions and allow for automatic event-based execution, e.g. when a user opens a file, saves a file, etc. This combination enables malevolent programmers to implement computer viruses²⁶.

A virus life-cycle goes something like the following. The virus itself is embedded within a word-processing document in the form of an executable macro. It spreads when users e-mail the document, download it from the web, etc. Opening the document usually triggers the virus via the macro language’s automatic execution feature. It then goes about doing the business of whatever

²⁵ Just download and install it — see <http://www.latex-project.org>. If you’re using the Unix[®] operating system, or any of the Unix[®]-like systems, e.g. Linux, you probably already have it on your computer. On the specific point of cost, open-source word-processors, such as OPENOFFICE (<http://www.openoffice.org>), do in fact compete effectively with L^AT_EX — they’re freely available too.

²⁶VISUAL BASIC FOR APPLICATIONS (VBA), which is the macro programming language in MICROSOFT OFFICE[®], is frequently used in the context of implementing viruses. VBA has been the *agent provocateur* of the most pandemic viruses, which have been spread mainly through the word-processor MICROSOFT WORD[®], the spreadsheet program MICROSOFT EXCEL[®], and the e-mail client MICROSOFT OUTLOOK[®]. See for example http://en.wikipedia.org/wiki/Computer_virus. Macro languages for more recent word-processors, e.g. OPENOFFICE (<http://www.openoffice.org>), exclude the automatic execution feature because of the security liability.

it was designed to do. This normally includes trying to copy itself to any other “infectable” files that can be found on a machine.

2.8 Learning Curve

The main objection I’ve heard regarding L^AT_EX is that it is too hard to learn. Complaints are usually leveled at the mark-up commands one uses to direct the typesetting process. For example, in L^AT_EX one specifies **boldface text** using the command `\textbf{boldface text}` in the input file²⁷. Although the typical person can get by with only having to know a few dozen such commands for much of their work, I am somewhat sympathetic to this problem.

What most people do not realize is that there are now well-established graphical interfaces to L^AT_EX, of which LYX is perhaps the most visible example²⁸. LYX has the “look and feel” of standard word-processor interfaces: it is basically menu-driven and includes spell checking, cut-and-paste capability, a mathematics editor, etc. *No knowledge of L^AT_EX is necessary in order to use it.* Such tools largely eliminate the perceived learning curve.

3 Conclusion

I mentioned at the outset that the issues we have just discussed were not given in any particular order. This is true to the extent that the relative importance of each of these will depend on your specific situation. In fact, you may have already ranked some of these for yourself in the course of reading.

In actuality, I ordered the points largely along my own preferences and priorities. The main advantages of typesetting for me personally revolve around the technical aspects in §2.1 through §2.4: versatility, typesetting capability, and longevity and portability of software and documents. The issue of document size in §2.5 is less important to me, mainly because I have network access, a very fast connection to the Internet, etc.

²⁷If using a word-processor, one selects (highlights) the desired words and presses the appropriate formatting button on the graphical user interface.

²⁸LYX can be obtained from <http://www.lyx.org>. Like the other free software tools we’ve discussed (e.g. footnote 25, pp. 13), it is free in the monetary sense.

The topic of monetary cost in §2.6 is an interesting one because of the general trend toward software commoditization²⁹. Whether one opts for typesetting or word-processing, there is no longer a need to *pay* for such software³⁰. I think this can be summed up by saying simply: if you're paying more than \$0.00 for office-type software to process text, you're probably paying too much.

The security aspect in §2.7 is likewise not as important to me personally because I do all my work on Unix[®] and Unix[®]-like operating systems, which are relatively secure. Users of the MICROSOFT WINDOWS[®] family of operating systems have to be much more vigilant in this regard. They have been the hapless victims of almost all virus-related security problems.

Lastly, the issue of the learning curve discussed in §2.8 is the least relevant in my case. I have always done some amount of programming in my work and am quite comfortable using a text editor and mark-up commands. Therefore, I use L^AT_EX in the “standard” way of creating a marked-up text file for direct input. However, as I pointed out, I no longer really consider this a substantive issue for *anyone* because very good graphical interfaces are now available. The perceived learning curve is one of the biggest stumbling blocks for word-processor users because most are unaware of front-end interfaces like LYX.

I shall close by simply saying I'm happy you read this far and am even happier if you now feel a little bit more informed on the issues surrounding document preparation. Try out the tools I've discussed here. You might be surprised to find that you have real (and better) alternatives to what you are currently using.

4 Document History

- version 1.0 (02-AUG-04): initial document implementation
- version 1.01 (16-AUG-04): minor wording corrections

²⁹Alot has been written about this. See e.g. “The Commoditization of Software” (2003) by John Carroll, available at <http://zdnet.com.com/2100-1107-992824.html>.

³⁰As mentioned in footnote 25 (pp. 13) there is abundant free software. I've found that most people still remain unaware of this fact. Free software is also starting to eliminate the necessity of add-on packages like ENDNOTE[®]. For example, OPENOFFICE (<http://www.openoffice.org>) has built-in capability for bibliographic databases.

References

- Baig, E. C., 2004. Life has gotten even shorter in the digital age. *USA Today* : July 26.
- Kopka, H. and Daly, P. W., 1999. *A Guide to Latex*. Addison–Wesley, Boston MA, 3rd edition.
- Toro, E. F., 1999. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer–Verlag, New York NY, 2nd edition.
- Wilke, C. O., 2004. Supplementary materials need the right format. *Nature* 430: 291–291.