1 Introduction

1.1 The Concept of Generic Markup

Originally markup was the annotation of manuscripts of a copy editor telling the typesetter how to format the manuscript. It consisted of handwritten notes such as '*set this heading in 12 point Helvetica italic on a 10 point body, justified on a 22 pica slug with indents of 1 em on the left and none on the right*.' With the advent of computers, these marks could be coded electronically using a special coding system and people started inventing their own markup schemes. The following low level formatting commands used to instruct a computer for *carriage return*, *center the following text*, and *go to the next page* are a typical example:

```
.pa ; .sp 2 ; .ce ; .bd
Title of the chapter
.sp
```

In another markup scheme it will be like as given hereunder:

```
\vfill\eject\begingroup\bf\obeylines\vskip 20pt
\hfil Title of the chapter
\vskip 10pt\endgroup\bigskip
```

Documents created with such specific markup became difficult for typesetting systems to cope up. A movement was started to create a standard markup language, which all typesetting vendors would be persuaded to accept as input. Thus came the Generic Markup Language (GML) which later on developed into Standard Generalized Markup Language (SGML) and now a subset of which known as Extensible Markup Language (XML) is poised to take up the World Wide Web.

However, the development of SGML moved towards representing the documents in an exchangeable format aiming at 'publishing in its broadest sense, ranging from single medium conventional publishing to multimedia data base publishing'. SGML can also be used in office document processing when the benefits of human readability and interchange with publishing systems as required. It is a meta-language for defining infinite variety of markup languages and is not concerned with the formatting of marked-up documents., *i.e.*, there is no layout tags.

This void is filled by the advent of TeX which combines the balance of generic markup and layout specific support. The class file mechanism followed in LaTeX makes it possible to produce the same source document in different layouts, while enough bells and whistles are available to fine-tune important documents for producing the highest quality.

1.2 A Short History of TeX

Donald E. Knuth

TeX (= tau epsilon chi, and pronounced similar to 'tech') is a computer language designed by Donald Erwin Knuth of Stanford University, for use in typesetting; in particular, for typesetting math and other technical (from Greek 'techne' = art/craft, the stem of 'technology') material.

In the late 1970s, Donald Knuth was revising the second volume of his multi-volume opus, *The Art of Computer Programming*, got the galleys, looked at them, and said (approximately) 'bleccch'! he had just received his first samples of the new computer typesetting,

and its quality was so far below that of the first edition of Volume 2 that he couldn't stand it. He thought for awhile, and said (approximately), "I'm a computer scientist; I ought to be able to do something about this", so he set out to learn what were the traditional rules for typesetting math, what constituted good typography, and (because the fonts of symbols that he needed really didn't exist) as much as he could about type design. He figured this would take about 6 months. (Ultimately, it took nearly 10 years, but along the way he had lots of help from some people who should be well known to readers of this list – Hermann Zapf, Chuck Bigelow, Kris Holmes, Matthew Carter and Richard Southall are acknowledged in the introduction to Volume E, *Computer Modern Typefaces*, of the Addison-Wesley *Computers & Typesetting* book series.)

A year or so after he started, Knuth was invited by the American Mathematical Society (AMS) to present one of the principal invited lectures at their annual meeting. This honor is awarded to significant academic researchers who (mostly) were trained as mathematicians, but who have done most of their work in not strictly mathematical areas (there are a number of physicists, astronomers, etc., in the annals of this lecture series as well as computer scientists); the lecturer can speak on any topic (s)he wishes, and Knuth decided to speak on computer science in the service of mathematics. The topic he presented was his new work on TEX (for typesetting) and METAFONT (for developing fonts for use with TEX). He presented not only the roots of the typographical concepts, but also the mathematical notions (e.g., the use of bezier splines to shape glyphs) on which these two programs are based. The programs sounded like they were just about ready to use, and quite a few mathematicians, including the chair of the Mathematical Society's board of trustees, decided to take a closer look. As it turned out, TEX was still a lot closer to a research project than to an industrial strength product, but there were certain attractive features:

- it was intended to be used directly by authors (and their secretaries) who are the ones who really know what they are writing about;
- it came from an academic source, and was intended to be available for no monetary fee (nobody said anything about how much support it was going to need);
- as things developed, it became available on just about any computer and operating system, and was designed specifically so that input files (files containing markup instructions; this is not a WYSIWYG system) would be portable, and would generate the same output on any system on which they were processed – same hyphenations, line breaks, page breaks, etc., etc.;
- other programs available at the time for mathematical composition were:
  - ⋆ proprietary
  - ⋆ very expensive
  - ⋆ often limited to specific hardware
  - ⋆ if WYSIWYG, the same expression in two places in the same document might very well not look the same, never mind look the same if processed on two different systems.

Mathematicians are traditionally, shall we say, frugal; their budgets have not been large (before computer algebra systems, pencils, paper, chalk and blackboards were the most important research tools). TEX came along just before the beginnings of the personal computer; although it was developed on one of the last of the 'academic' mainframes (the DECsystem (Edusystem)-10 and -20), it was very quickly ported to some early HP workstations and, as they emerged, the new personal systems. From the start, it has been popular among mathematicians, physicists, astrophysicists, astronomers, any research scientists who were plagued by lack of the necessary symbols on typewriters and who wanted a more professional look to their preprints.

To produce his own books, Knuth had to tackle all the paraphernalia of academic publishing—footnotes, floating insertions (figures and tables), etc. As a mathematician/computer scientist, he developed an input language that makes sense to other scientists, and for

math expressions, is quite similar to how one mathematician would recite a string of notation to another on the telephone. The TeX language is an interpreter. It accepts mixed commands and data. The command language is very low level (skip so much space, change to font X, set this string of words in paragraph form, ...), but is amenable to being enhanced by defining macro commands to build a very high level user interface (this is the title, this is the author, use them to set a title page according to AMS specifications). The handling of footnotes and similar structures are so well behaved that 'style files' have been created for TeX to process critical editions and legal tomes. It is also (after some highly useful enhancements in about 1990) able to handle the composition of many different languages according to their own traditional rules, and is for this reason (as well as for the low cost), quite widely used in eastern Europe.

Some of the algorithms in TeX have not been bettered in any of the composition tools devised in the years since TeX appeared. The most obvious example is the paragraph breaking: text is considered a full paragraph at a time, not line-by-line; this is the basic starting algorithm used in the HZ-program by Peter Karow (and named for Hermann Zapf, who developed the special fonts this program needs to improve on the basics).

In summary, TeX is a special-purpose programming language that is the centerpiece of a typesetting system that produces publication quality mathematics (and surrounding text), available to and usable by individuals.

1.3 What is LaTeX then?

At the beginning of 1980s, Leslie Lamport started work on a document preparation system called LaTeX based on the TeX formatter. This system adds a set of functions that makes the TeX language more friendlier than using the primitives provided in TeX, enabling the author to concentrate on the content and structure of the document rather than the formatting details, so that the author might not loose the train of thought while writing his document. Also, LaTeX's functionality, in conjunction with a few auxiliary programs, includes the generation of indices, bibliographies, cross-references, tables of contents, graphic inclusion, etc. These are the features that are lacking in basic TeX (usually called plain TeX).

1.4 Getting Started

First of all, let's see what steps are necessary to produce a document using LaTeX. The first step is to type the file that LaTeX reads. This is usually called the LaTeX file or the input file, and it can be created using a simple text editor (in fact, if you're using a fancy word processor, you have to be sure that your file is saved in ASCII or non-document mode without any special control characters). The LaTeX program then reads your input file and produces what is called a DVI file (DVI stands for DeVice Independent). This file is not readable, at least not by humans. The DVI file is then read by another program (called a device driver) that produces the output that is readable by humans. Why the extra file? The same DVI file can be read by different device drivers to produce output on a dot matrix printer, a laser printer, a screen viewer, or a phototypesetter. Once you have produced a DVI file that gives the right output on, say, a screen viewer, you can be assured that you will get exactly the same output on a laser printer without running the LaTeX program again.

The process may be thought of as given in the Figure 1.1. This means that we don't see our output in its final form when it is being typed at the terminal. But in this case a little patience is amply rewarded, for a large number of symbols not available in most word processing programs become available. In addition, the typesetting is done with more precision, and the input files are easily sent between different computers by electronic mail or on a magnetic medium.

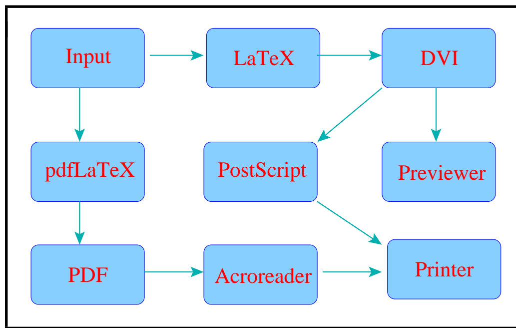

Figure 1.1 The LaTeX production chain

```
\documentclass[a4paper]{tutorial}
\pagestyle{headings}
\usepackage[screen,rightpanel,paneltoc,code]{pdfscreen}

\begin{document}

\chapter{Introduction}

\section{The Concept of Generic Markup}
Originally markup was the annotation of manuscripts
of a copy editor telling the typesetter how to format
the manuscript. It consisted of handwritten notes such
as `\emph{set this heading in 12 point Helvetica}
```

Figure 1.2 A sample LaTeX input file

Our focus will be on the first step, that is, creating the LaTeX input file and then running the LaTeX program to produce appropriate results. There are two ways of running the LaTeX program; it can be run in batch mode or interactively. In batch mode you submit your LaTeX input file to your computer; it then runs the LaTeX program without further intervention and gives you the result when it is finished. In interactive mode the program can stop and get further input from the user, that is, the user can interact with the program. Using LaTeX interactively allows some errors to be corrected by the user, while the LaTeX program makes the corrections in batch mode as best it can. Interactive is the preferred mode, of course. All personal computer and many mainframe implementations are interactive. On some mainframes, however, the only practical method of running LaTeX is in batch mode.

1.4.1 A typical LaTeX input file

The preamble portion of a LaTeX input file that generated the *Introduction* page of this document is given in Figure 1.2. You will notice that there are many keywords that start with the character '\' followed by arguments within '`[ ]`' and '`{ }`'. These keywords are called *control sequences*, the arguments within square brackets are called optional arguments and those within curly braces are called arguments (which are mandatory). We will learn about these later on.

When you run LaTeX over this file (for the time being, we shall name it as `test.tex`), we get the output called `test.dvi`. The `web2c` TeX system is the implementation distributed by the TeX Users Group and is free. Throughout this tutorial, we shall describe TeX functionality based on `web2c` system only. Commercial implementations like PCTeX and Y&YTeX for Win32 systems or *Textures* for Macintosh, though widely used in the typesetting industry, will not be described in this manual owing to its non-GNU nature.

You can issue the following command to the command prompt of your Unix shell to compile your input file (here we call `test.tex`):

```
$ latex test
```

Extension is only necessary, if you have given extension other than `*.tex`. In Win32 system, you can use the TEXshell and can click at the LATEX button to run LATEX.

Many previewers are available, `xdvi` is the standard previewer in Unix and `Windvi` in Win32 systems. The following command will show your dvi in your computer screen. Again, extension is only optional.

```
$ xdvi test
```

Printing is usually done through POSTSCRIPT. You can convert the `dvi` into `ps` by issuing the following command:

```
$ dvips test -o test.ps
$ lpr test.ps
```

This will print the dvi to your printer. `dvips` can be configured to pipe the `*.ps` directly to your printer. Win32 systems provide menu buttons to accomplish these jobs.

You might be amused to know that this `test.dvi` is independent of any platform and devices. You can view this output in any dvi previewer of any operating system irrespective of the OS of origination and can be printed in any printer for the identical output, which is not the case with the WYSIWYG typesetting systems that are usually hard wired to the installed printer, the format changes as soon as you change your printer. Therefore, TEX is device and platform independent. Also you can compile the very same TEX sources in any TEX system in any operating system irrespective of its originating OS. This platform independence has made TEX documents a choice of transfer, especially scientific documents over the INTERNET.