Producing On-line Information Files with LaTeX

Hubert Partl
Technische Universität Wien

Printed Documents and On-line Information Files

Computer users nowadays expect that all necessary information about the computer is available in two forms: as printed documents, and as on-line information files.

- The printed documents are usually bought at the EDP center's book shop. They may be anything from short leaflets to complete books, and in any case they should be as beautiful and as readable as possible. Thus, TeX, together with one of its macro packages like LaTeX, and a laser printer are the ideal means to generate them.
- On-line information files are ASCII files stored on the computer itself. Whilst working on the computer, users can access them directly from the terminals on their own desks. Typically, the files are viewed on a terminal screen, or searched for certain keywords with an editor, or printed on a nearby cheap line printer.

For the authors and maintainers of the information texts, it is highly desirable that the same input file can be used to generate both the printed and the on-line versions. How can this be achieved in a LaTeX environment?

There are driver programs available that generate line printer or ASCII output from a DVI file, but these are aimed at proof-reading and previewing: They try hard to show how the text will eventually be broken into lines and pages in the final printed document. Their results as such are usually neither beautiful nor readable.

What we need is something different: We want the text to be set as beautiful and as readable as is possible in a simple line-oriented ASCII file, and with a layout that is best suited for the purpose of viewing on a terminal and printing on a line printer (e.g. 72 characters per line, 60 lines per page, blank lines to separate sections, and so on). We don't care about any relation to the line and page breaks of the printed version.

Obviously, we need two things to achieve this:

1. We need a LaTeX style or style option that will set our text such that the ASCII file will be as readable and as beautiful as possible.
   This will be dealt with in the present article.
2. We need an ASCII driver that will convert our DVI file into an ASCII file.
   Several such drivers are available, most of them in the public domain. My favorite is Crudetype by R. M. Damerell (Royal Holloway and Bedford College, Egham, UK).

My First Attempt

Here is my first attempt for such a document style option file, which I called `screen.sty`.

It is intended for generating ASCII file versions of non-mathematical texts, usually descriptions of computer programs and similar information.

Ideally, it should work like this:

- The printed manual is generated the usual way, with LaTeX and with a laser printer driver.
- The on-line version is generated by using a copy of the LaTeX input file, in which the option `screen` is added at the end of the option list in the `\documentstyle` command, and by feeding the resulting DVI file into an ASCII driver like Crudetype.

Real life, however, is a bit more complicated: I usually have to apply several manual changes to the copied version of the LaTeX input file and also to the generated output file. But even so, this is much easier than maintaining two completely different text files for the printed and on-line versions.

In spite of these drawbacks, I am presenting my humble first approach to the TUGBOAT readership — hoping that some readers can use my ideas, and that some will provide me with their ideas of how to do it better!

Now, let us have a look at the contents of my `screen.sty` file:

Fonts

Only one "font" is available on a line printer, and it is a mono-spaced one. With TeX, this means that the whole document should be set in the typewriter font (`\tt`).

The normal-size typewriter font is selected by the following commands:

```
\normalsize\tt
```

This must be done at the beginning, so that `\baselineskip` and the `em` and `ex` units have the correct values for all length assignments to come.

All font changing commands are re-defined to refer to the `\tt` font:

```
\let\rm=\tt  \let\bf=\tt  \let\it=\tt
\let\sl=\tt  \let\sf=\tt  \let\sc=\tt
\let\em=\tt
```

Of course, this does not catch special fonts loaded by the user, nor implicit font changes other than `\em`, nor does it consider the mathematical mode. If the LaTeX input file contains commands to load special fonts, these should be `\let` to `\tt`, too.

Font Sizes

Only one character size is available on a line printer. All size changing commands are changed to refer to `\normalsize`, and instead of switching to the `\rm` font, they will switch to the `\tt` font.

First, we change the `\normalsize` command to switch to `\tt` rather than `\rm`:

```
\let\NormalsizeRm=\normalsize
\def\normalsize{\NormalsizeRm\tt}
```

Then, all other size changing commands are re-defined to this new `\normalsize` command:

```
\let\tiny=\normalsize
\let\scriptsize=\normalsize
\let\footnotesize=\normalsize
\let\small=\normalsize
\let\large=\normalsize
\let\Large=\normalsize
\let\LARGE=\normalsize
\let\huge=\normalsize
\let\Huge=\normalsize
```

Accents and Special Characters

Overprinting should be avoided in the ASCII file generated, because it does not work when the file is viewed on a terminal screen. All accented and special characters have to be mapped to "normal" ASCII characters or character sequences.

Most accents can just be omitted, i.e. é can be printed as e, and so on.

```
\let\'=\relax \let\`=\relax \let\^=\relax
\let\c=\relax \let\~=\relax \let\==\relax
\let\.=\relax \let\u=\relax \let\v=\relax
\let\H=\relax \let\d=\relax \let\b=\relax
\let\t=\relax
```

Various umlaut characters are to be replaced by two-character sequences, e.g. ä will be printed as ae, ß will be printed as ss, œ will be printed as oe, and so on. Most of these changes are straightforward:

```
\def\ss{ss}  \def\aa{aa}  \def\ae{ae}
\def\oe{oe}  \def\AA{Aa}  \def\AE{Ae}
\def\OE{Oe}  \def\o{oe}   \def\O{Oe}
```

However, with the umlaut accent `\"`, the following has to be considered: The german umlaut characters ä, ö, and ü are to be printed as ae, oe, and ue, but with the other letters, the umlaut dots can just be omitted, e.g. oë can be printed as oe, and aï can be printed as ai. Therefore, the `\"` command is re-defined like this:

```
\def\"#1{\ifmmode #1\else
        \if \string #1aae\else
        \if \string #1ooe\else
        \if \string #1uue\else
        \if \string #1AAe\else
        \if \string #1OOe\else
        \if \string #1UUe\else
        #1\fi \fi \fi \fi \fi \fi \fi}
```

Since we do not set any accents above letters, the dotless i and j can be replaced by their normal dotted versions:

```
\def\i{i}          \def\j{j}
```

Language Specific Modifications

Non-English speaking LaTeX users may use some modified versions of the LaTeX document styles. For instance, the `german` style option is likely to be used for german texts. In this case, some of the language specific definitions may need similar re-definitions.

If the option file `german.sty` has been processed, the active quotes character " must be re-defined with respect to the umlaut characters, the sharp s and the german quotes. The `\@ifundefined` command can be used to test whether the `german` option has been specified.

Here is an example how the french quotes ("guillemets") can be re-defined:

```
\def\flqq{{\tt <<}}    \def\frqq{{\tt >>}}
```

Mathematical Symbols

We do not consider the typesetting of mathematical formulae. However, even in non-mathematical texts, some symbols of TeX's math mode are used, e.g. dots, bullets, and arrows. These commands are re-defined to print appropriate ASCII characters or character sequences, switching to text mode and to the `\tt` font:

```
\def\ldots{\mbox{\tt ...}}
\let\cdots=\ldots
\let\dots=\ldots
\def\times{\mbox{\tt x}}
\def\bullet{\mbox{\tt *}}
\def\rightarrow{\mbox{\tt ->}}
\def\Rightarrow{\mbox{\tt =>}}
\def\longrightarrow{\mbox{\tt -->}}
\def\Longrightarrow{\mbox{\tt ==>}}
\def\leftarrow{\mbox{\tt <-}}
\def\Leftarrow{\mbox{\tt <=}}
\def\longleftarrow{\mbox{\tt <--}}
\def\Longleftarrow{\mbox{\tt <==}}
```

Other symbols should be added to this list, if needed.

Other Special Characters

Two conversion problems still need to be solved:

In normal text, the character sequences `--` and `---` produce long dashes. Since we have switched to the `\tt` font, however, they produce two or three hyphens, respectively. It would be nice to find an automatic way that makes the character sequences `--` and `---` print a single hyphen (`-`) only.

A similar problem exists for the opening and closing quotes: In normal text, ` `` ` and `''` are used to print opening and closing quotes. Since we have switched to the `\tt` font, they produce double apostrophes. Instead, we would like them to print one double quotes character (`"`).

I have not found a suitable definition yet that would accomplish this within TEX. I assume that it should involve `\catcode` to make the characters active, and `\@ifnextchar` to look at the following character. However, the hyphen sign and quotes should not become "fragile", and the re-definitions should not inhibit the use of hyphens within the `\hyphenation` command nor the use of the grave character within the `\catcode` command...

Perhaps, a better way would be to define ligatures for verb———, ` `` ` and `''` in the TFM-File for font `cmtt10`. Of course, these ligatures should be disabled in the `verbatim` mode, i.e. they should be included in the `\@noligs` command.

Kerning, Raising and Lowering

Both the character positions within each line and the line positions within each page are fixed in the line printer file. Therefore, kerning, raising, and lowering of characters must be avoided.

The following re-definitions make the `_` and `^` commands do nothing in mathematical mode:

```
\catcode`\_=\active  \let_=\relax
\catcode`\^=\active  \let^=\relax
```

The following re-definitions generate appropriate substitutes for the TEX and LATEX logos:

```
\def\TeX{TeX}      \def\LaTeX{LaTeX}
```

Similar re-definitions should be added for other logos of this kind, if needed.

Vertical Skips

The files to be generated consist of discrete lines (as opposed to arbitrary character placement on the page). Therefore, all vertical skips must be integer multiples of the line height `\baselineskip`, and they must not be stretched or shrunk.

The `\baselinestretch` factor must be 1:

```
\def\baselinestretch{1}
```

If paragraphs are indented with no vertical skip, `\parskip` can be set to zero:

```
\parskip=0pt
```

If, however, they are not indented but is to be separated by a vertical skip, this skip should be one blank line:

```
\parskip=\baselineskip
```

The predefined vertical skips are re-defined to zero or one line, respectively:

```
\smallskipamount=0pt
\medskipamount=\baselineskip
\bigskipamount=\baselineskip
```

The sectioning commands are re-defined such that the vertical skips and their stretching are multiples of the line height, and that the heading is printed in `\tt` style:

```
\def\section{\@startsection
    {section}{1}{\z@}%
    {-2\baselineskip plus -2\baselineskip}%
    {1\baselineskip}%
    {\raggedright\normalsize\tt}}
\def\subsection{\@startsection
    {subsection}{2}{\z@}%
    {-1\baselineskip plus -1\baselineskip}%
    {1\baselineskip}%
    {\raggedright\normalsize\tt}}
\def\subsubsection{\@startsection
    {subsubsection}{3}{\z@}%
    {-1\baselineskip plus -1\baselineskip}%
    {1\baselineskip}%
    {\raggedright\normalsize\tt}}
```

Note that due to the `\raggedbottom` command (see below), the stretchable glue in these skips will *not* cause an actual stretching but will help LATEX to find a suitable place for the page breaks.

The vertical skips used in all list environments are re-defined, too. Here is an example that re-defines them all to equal `\parskip`:

```
\def\@listI{\leftmargin\leftmargini
    \topsep\z@ \parsep\parskip
    \itemsep\z@}
\let\@listi\@listI
\@listi
\def\@listii{\leftmargin\leftmarginii
    \labelwidth\leftmarginii
    \advance\labelwidth-\labelsep
    \topsep\z@ \parsep\parskip
    \itemsep\z@}
\def\@listiii{\leftmargin\leftmarginiii
    \labelwidth\leftmarginiii
    \advance\labelwidth-\labelsep
    \topsep\z@ \parsep\parskip
    \itemsep\z@}
```

Of course, these re-definitions don't catch explicit `\vspace` and `\\[`*length*`]` commands that appear in the LaTeX input file. These may have to be changed manually.

The \raggedbottom command makes sure that vertical skips are never stretched:

```
\raggedbottom
```

Horizontal Skips

In an ASCII file, horizontal skips can only be accomplished by space characters which all have the fixed character width. Therefore, all horizontal skips must be multiples of the \tt font's character width. Note that 1 em is 2 character widths in this font.

If paragraphs are indented, \parindent should be set to something like:

```
\parindent=1em
```

If, however, they are not indented but are separated by a vertical skip only, \parindent is set to zero:

```
\parindent=0pt
```

The indentation amounts of all list environments are re-defined like this:

```
\leftmargini=2em
\leftmargin=\leftmargini
\leftmarginii=2em
\leftmarginiii=2em
\leftmarginiv=2em
```

The dot distance in the dotted lines within the table of contents has to be re-defined, too:

```
\def\@dottedtocline#1#2#3#4#5{%
  \ifnum #1>\c@tocdepth \else
  \vskip \z@ plus .2pt
  {\leftskip #2\relax \rightskip\@tocrmarg
   \parfillskip -\rightskip
   \parindent #2\relax\@afterindenttrue
   \interlinepenalty\@M
   \leavevmode
   \@tempdima #3\relax
   \advance\leftskip \@tempdima
   \hbox{}\hskip -\leftskip
   #4\nobreak\leaders\hbox{\tt ~.~}\hfill
   \nobreak \hbox to\@pnumwidth
           {\hfil\rm #5}\par}\fi}
```

(This differs from the original definition only in the argument of the \leaders command.)

Of course, these re-definitions don't catch explicit \hspace or \kern commands that appear in the LaTeX input file. These may have to be changed manually.

The \raggedright command is needed, because in the \tt font, the spaces have fixed width and cannot be stretched for justification:

```
\raggedright
```

Page Layout

The line width is set to 72 characters per line:

```
\textwidth=36em
```

The text height is set to 54 lines, which corresponds to 9 inches if the print density is 6 lines per inch:

```
\textheight=54\baselineskip
```

Other values (e.g. 80 characters per line) may be chosen in a similar way.

The top margin (including the header area) is set to zero:

```
\topmargin=0pt
\advance \topmargin by -\headheight
\advance \topmargin by -\headsep
```

The left margin, too, is set to zero for all pages:

```
\oddsidemargin=0pt
\evensidemargin=\oddsidemargin
```

The empty pagestyle is selected, because page numbers are normally not printed in on-line information files:

```
\pagestyle{empty}
```

However, for long documents, page numbers may be desirable. This can be accomplished by issuing an appropriate \pagestyle command in the LaTeX input file.

Future Work

I am well aware that what I have presented here is a very first attempt only. Several extensions are certainly necessary to make it generally applicable. For instance, I have included the most simple text elements and environments only, but did not consider many other issues like footnotes, marginal notes, rules, tables, figures, title pages, abstracts, bibliographies, and so on. And I have completely omitted mathematics and pictures.

Furthermore, my re-definitions may contain bugs and errors, and perhaps a completely different approach would be much better.

It is my hope that many TeXperts will now go on with this topic and propose and discuss different approaches — by e-mail, via the TeXhax mailing list, and in future editions of TUGBOAT. I am looking forward to many articles titled *"Another approach to producing on-line information files with LaTeX"*.

◇ Hubert Partl
EDV-Zentrum
Technische Universität Wien
Wiedner Hauptstraße 8–10
A-1040 Wien, Austria
Bitnet: z3000pa@awituw01