Bijlage 4
Toolbox: let's keep things plain

Maarten Gelderman

abstract
This Toolbox follows the eclectic approach that most readers will know from previous ones. Without aiming to give a comprehensive oversight of the logic behind them, I present some plain TEX-commands that can be used directly in LATEX. As the editors of this journal decided to make English the preferred language for contributions, this column is no longer in Dutch. However, as far as my limited command of the language allows, I will try to keep the tone informal and the discussion accessible to novice users.

Keywords: plain TEX in LATEX

Some background

Most readers of this journal will be familiar with the fact that TEX and LATEX are not the same thing. TEX, the program is the 'machine' that runs LATEX. LATEX, the macro-package, is the system that is run by most of us in order to process our documents. Instead of LATEX, we may run any other macro package. The most well-known are Context, eplain, and plain. Context is a monolithic system which instantaneously provides functionality LATEX can only offer if combined with a whole bunch of packages. Plain is the original TEX macro-package developed by Donald Knuth and eplain is an extension of plain that provides some LATEX-like functionality without sacrificing any of the original plain possibilities

The last sentence already indicates that other macro-systems may make some plain-TEX functionality inaccessible, and indeed this is the case. LATEX-makes some of the plain TEX macro's unavailable or unusable. Fortunately, the majority of those commands still works and LATEX-users may well use them to their advantage. This column will sketch the possibilities of some plain-TEX macro's. If will mainly focus on some commands and parameters that influence low-level formatting. The information in this column is based on documentation in `ltplain.dtx` and the TEX-book (especially Chapter 14).

Line breaking and overfull boxes

One of the most advantageous areas for application of plain macro's is in manipulating hyphenation and justification. As long as you stick to the computer modern fonts most of the standard settings work just fine. However, when you switch to some other font (say "times" by e.g. issuing the command `\usepackage{times}` or `\rmdefault{ptm}`) you may almost feel drowned by the amount of overfull boxes. Of course, you will finally get rid of those by carefully checking all hyphenation possibilities. However, while on a document this is too much work and the overfull boxes can become a real nuisance. One possible measure is to use LATEX's `sloppypar`-environment. However, this will affect the look of your document rather negatively. A more subtle correction can be made by `\emergencystretch`. Emergencystretch is the amount by which the spaces in a single line may be enlarged in order to avoid overfull boxes when the ordinary line-breaking algorithm fails. By issuing `\emergencystretch=1em` you should be able to get rid of the majority of those annoying messages.

Another way to avoid messages without improving your document is adapting the value of `\tolerance`. TEX will only produce overfull boxes if it finds no way to break a paragraph into lines without creating ugly lines like the ones you just have been reading. In order to determine whether a line is too ugly to be acceptable, TEX compares the ugliness of a line with the value of `\tolerance`. Standard LATEX sets tolerance at 200, by setting e.g. `\tolerance=400` uglier lines (more like the ones commercial word-processors produce) are permitted and fewer overfull boxes will be produced.

A final small trick that may save you the trouble of manually correcting a lot of bad hyphenation is `\slash`. It is simply used to produce a slash (/) that functions like a hyphen. Hence "automatisering`\slash` mechanisering" will be hyphenated as "automatisering/ mechanisering", if necessary (I forced it to be necessary here by adding yet another `\break`). You may make commands like `\slash` yourself. Just use `\def\en{--\discretionary{}{}{}}` to get xxxxxxxxxxxx–xxxxxxxxxxxxx.

Changing the length of paragraphs

In the previous section we were concerned about the ease with which acceptable output can be produced. This sec-

Table 1. Automatically generating a font-inventory.

```
\input multido
{\font\test pzdr at 10pt \multido{\i=0+1}{255}
{ \i: {\test\char\i},}\ 255: \char255}.
```

0: , 1: , 2: , 3: , 4: , 5: , 6: , 7: , 8: , 9: , 10: , 11: , 12: , 13: , 14: , 15: , 16: , 17: , 18: , 19: , 20: , 21: , 22: , 23: , 24: , 25: , 26: , 27: , 28: , 29: , 30: , 31: , 32: , 33: ✁, 34: ✂, 35: ✃, 36: ✄, 37: ☎, 38: ✆, 39: ✇, 40: ✈, 41: ✉, 42: ☛, 43: ☞, 44: ✌, 45: ✍, 46: ✎, 47: ✏, 48: ✐, 49: ✑, 50: ✒, 51: ✓, 52: ✔, 53: ✕, 54: ✖, 55: ✗, 56: ✘, 57: ✙, 58: ✚, 59: ✛, 60: ✜, 61: ✝, 62: ✞, 63: ✟, 64: ✠, 65: ✡, 66: ✢, 67: ✣, 68: ✤, 69: ✥, 70: ✦, 71: ✧, 72: ★, 73: ✩, 74: ✪, 75: ✫, 76: ✬, 77: ✭, 78: ✮, 79: ✯, 80: ✰, 81: ✱, 82: ✲, 83: ✳, 84: ✴, 85: ✵, 86: ✶, 87: ✷, 88: ✸, 89: ✹, 90: ✺, 91: ✻, 92: ✼, 93: ✽, 94: ✾, 95: ✿, 96: ❀, 97: ❁, 98: ❂, 99: ❃, 100: ❄, 101: ❅, 102: ❆, 103: ❇, 104: ❈, 105: ❉, 106: ❊, 107: ❋, 108: ●, 109: ❍, 110: ■, 111: ❏, 112: ❐, 113: ❑, 114: ❒, 115: ▲, 116: ▼, 117: ◆, 118: ❖, 119: ◗, 120: ❘, 121: ❙, 122: ❚, 123: ❛, 124: ❜, 125: ❝, 126: ❞, 127: , 128: , 129: , 130: , 131: , 132: , 133: , 134: , 135: , 136: , 137: , 138: , 139: , 140: , 141: , 142: , 143: , 144: , 145: , 146: , 147: , 148: , 149: , 150: , 151: , 152: , 153: , 154: , 155: , 156: , 157: , 158: , 159: , 160: , 161: ❡, 162: ❢, 163: ❣, 164: ❤, 165: ❥, 166: ❦, 167: ❧, 168: ♣, 169: ♦, 170: ♥, 171: ♠, 172: ①, 173: ②, 174: ③, 175: ④, 176: ⑤, 177: ⑥, 178: ⑦, 179: ⑧, 180: ⑨, 181: ⑩, 182: ❶, 183: ❷, 184: ❸, 185: ❹, 186: ❺, 187: ❻, 188: ❼, 189: ❽, 190: ❾, 191: ❿, 192: ➀, 193: ➁, 194: ➂, 195: ➃, 196: ➄, 197: ➅, 198: ➆, 199: ➇, 200: ➈, 201: ➉, 202: ➊, 203: ➋, 204: ➌, 205: ➍, 206: ➎, 207: ➏, 208: ➐, 209: ➑, 210: ➒, 211: ➓, 212: ➔, 213: →, 214: ↔, 215: ↕, 216: ➘, 217: ➙, 218: ➚, 219: ➛, 220: ➜, 221: ➝, 222: ➞, 223: ➟, 224: ➠, 225: ➡, 226: ➢, 227: ➣, 228: ➤, 229: ➥, 230: ➦, 231: ➧, 232: ➨, 233: ➩, 234: ➪, 235: ➫, 236: ➬, 237: ➭, 238: ➮, 239: ➯, 240: , 241: ➱, 242: ➲, 243: ➳, 244: ➴, 245: ➵, 246: ➶, 247: ➷, 248: ➸, 249: ➹, 250: ➺, 251: ➻, 252: ➼, 253: ➽, 254: ➾, 255: .

tion takes the opposite approach. In the final stage of document preparation fine-tuning of the appearance of pages will be required. On a global level you may for instance want to influence the length of your document. When a book is printed, 16 pages are processed at a time. If the length of the text you are working on is 194 rather that 192 pages, sixteen additonal pages have to be printed of which only two will be used. You may try to reduce the length of your text by setting `\linepenalty=`*nnn* (where *nnn* is a number between -10000 en 10000). LaTeX sets this parameter at 10. By e.g. raising it to 100 you discourage the line-breaking algorithm of TeX from making additional lines and hence may be able to reduce the overall length of your printout.

In some cases it may be desirable to manipulate the length of individual paragraphs. You can use `\looseness` to achieve this. If you set `\looseness=1` TeX will try to make the paragraph one line longer than it would otherwise do. If you use a negative value, paragraphs will be shortened. In this way widows and orphans can be eliminated manually (see also the final paragraph of this section).

On a micro level, footnotes may cause problems. If you typeset a book with LaTeX, and this book contains footnotes, almost inevitably some of them will be split across pages. Often for, to the human mind, no good reason at all. Setting `\interlinepenalty=10000` *locally* in the footnote, will assign infinite badness to such breaking and hence will keep the footnotetext together (if you set this variable *globally* not a single paragraph in your document will be split across pages, which may not be exactly what you want). It is not necessary to set the value to infinite (10000) you may choose any large value in order to discourage breaking of footnotes (and paragraphs in general) across pages.

Similar to the footnote-problem is the occurance of widows and orphans (when the first line of a paragraph is placed on another page than the remainder, this line is called a widow, if the same accident occurs to the last line, it is called an orphan). By setting `\widowpenalty` and `\clubpenalty` to a large value, such behaviour is discouraged. LaTeX fixes both values at 150.

Easy font changes

The subject matter discussed above is mainly concerned with low level formatting that cannot be done from LaTeX directly. The issue presented in this section, font selection, can be done rather adequately by LaTeX itself. However, in some cases the LaTeX-approach is just too heavy. Take the case where we have a font file from which we want to use just a single character (this may be a dingbats font—a font with symbols rather than letters—or a Metafont-file generated by GnuPlot). Setting up such a font using the New Font Selection Scheme (NFSS) requires that you generate font definition files. The plain TeX approach is far more simple: you just need your font and the associated TFM-file.[1] For the Zapf DingBats font, the name of the TFM-file is `pzdr.tfm`. Issue the command `\font\test pzdr at 10pt` (omitting the `.tfm` part of the file name)

1. TFM-files for PostScript font can be generated by the utility AFM2TFM.

to load this font at the required size (10 points in this case) and the font is ready for use. `\test 1234567890` will generate the following output: ✑✒✓✔✕✖✗✘✙✐. A small example of the beautiful Zapf Changery font is produced just as easily: look up the name of the TFM-file involved in the directory `/texmf/fonts/tfm/adobe/zapfding` (it turns out to be pzcmi) and enter: `\font\test pzcmi at 10 pt \test This is a beautiful Hamburgefont, isn's it?` which produces *This is a beautiful Hamburgefont, isn's it?*

Multido

In order to be able to use a DingBats font, you will need to find out where each symbol can be found. You may use the individual symbols by their corresponding letters (as was done in the example above), a more straightforward approach, however, is to access the individual symbols by the command `\char`*nnn*, where *nnn* is a number between 0 and 255 (a single TFM-file cannot contain more than 256 glyphs). Manually trying out all possibilities wouldn't be too exiting of course. Fortunately the (plain) TEX-package multido is available. Just load it, and let TEX do the work (see Table 1). A 'translation' of the commands in Table 1 would read: input the file `multido.tex`, load the TFM-file `pzdr` at a size of 10 points and next start repeating the command $i = i + 1$ 255 times, each time executing the third argument of the `\multido`-command, which tells TEX to print `\i` and typeset character i from the selected font. After finishing the iterations, print 255 and the corresponding character.

A warning

In applying the suggestion mentioned above be warned that most documents simply are not suited to be perfectly formatted. TEX will search for the best solution it can find, given the badnesses you provide it with. However, most times you will be trading of one badness against another; simply fixing all penalties and demerits at infinite will most probably result in an ugly document.