Tuning LaTeX to one's own needs

Jacek Kmiecik

Abstract

I will not conceal that the topic brought up in the article has been discussed many times over, especially on mailing lists — but it continues to come back. There are numerous methods of adjusting LaTeX to particular typographic needs.

I am not going to advocate for a particular, the only proper, way of tackling low-level modifications of LaTeX macros, but will be suggesting several ways of handling such situations. As usual, there are many ways. The choice depends on how well one masters the tools. Also, I will restrict myself to selected areas of typesetting, those where adjustments are most often needed.

1 Introduction

LaTeX is for many the first form of contact with a TeX environment. In many cases, it remains so and LaTeX becomes the only set of macros (classes, styles, packages) which a user is able to work with and which he trusts. His own macro-creation is limited to tiny modifications of basic definitions as described either in textbooks, or in easily accessible documentation.

A more demanding necessity might make a user search for an appropriate macro package in the cavernous CTAN archives. According to various "omniscient" authorities and mailing lists, these packages are supposed to provide — magically — a wonderful automatic mechanism of improving everything simultaneously. If it is not a single package of macros, then, unfortunately, other packages — supplementing successive missing elements of the typesetting — are being added to the LaTeX preamble.

However, sooner or later it turns out that adding one more package breaks the compilation, changes the hitherto acceptable typesetting in impermissible places, or warns with announcements about some incompatibility.

Thanks to such situations, we can often encounter statements similar to the following: "I don't use LaTeX, because once something didn't work as I wanted (...) and that's why I use Plain TeX".

That's right! This should be the next step in one's TeX education. Nevertheless, writing all macros for a voluminous publication may become a spectacular challenge — some functionalities can be programmed quickly and easily (e.g., headings, imprints, footnotes), others demand advanced knowledge of TeX (the contents, preparation of a hyperlinked PDF file, multiple runs, etc.). Undisputed educational and cognitive values do not often go hand in hand with the hurry with which we have to cope with difficult cases, hence there is an irresistible temptation to use LaTeX as it seems to be an easier and friendlier environment. At least, it was the assumption of its creator, Leslie Lamport.

And if we ... let's say ... "marry" Plain TeX with LaTeX? In principle, all LaTeX macros are more or less extended plain macro definitions. Instead of being exposed to limitations of ready-made packages, it suffices to redefine the most necessary LaTeX commands for our own local use. Any way we look at it, the majority of packages originated in this way — locally needed definitions, modified canonic LaTeX macros — all have been compiled into a separate package and adjusted to be used with the contemporary LaTeX 2$_\varepsilon$ format. The issue of maintaining compatibility is a different topic, which we are not going to elaborate on. Nonetheless, we should be aware of the existing dangers.

The purpose of this article is to suggest ways of modifying of some basic, canonical LaTeX definitions, starting from their source code.

2 What should we start from?

From the documentation! The LaTeX 2$_\varepsilon$ standard distribution contains quite well-documented source code — in your basic installation you can find for sure the source directory, and in it a file entitled `source2e.tex`. Its compilation gives $\approx$600 pages of documentation of the LaTeX 2$_\varepsilon$ format source code — makes for good bedtime reading! Very instructive! And how "plain" it is! Almost everything is written with the conventions of primary plain macros.

3 And what's next?

Apply the method of small steps. If your document requires the use of colors and the desired destination format of the publication is PDF — I do not recommend that anyone write a whole colourful PDF machinery from scratch. We have verified and tested packages `color.sty` or `xcolor.sty`, so we do not have to reinvent the wheel. Likewise for the graphics embedding macros or building tabular setups.

Let's suppose we have to deal with a voluminous publication. As a first step I would suggest dividing the document into logically smaller parts, e.g., chapters, and place them in separate files attached to the main document file. In this file, in the preamble, we can put the additional essential commands,

This is a translation of the article "Dostosowanie LaTeX-a do konkretnych potrzeb", which first appeared in *Biuletyn GUST* 25 (2008), pp. 44–52. Reprinted with permission. Translation by Jerzy Ludwichowski and the author.

or — what seems to me the most appropriate solution — we can create our own package or even a class. Building our own local macro package seems to be decidedly easier for a TEX beginner, so we will choose this particular solution. Our own file `mystyle.sty` should be incorporated into the preamble as follows:

```
\usepackage{mystyle}
```

Such a solution allows the use of any plain or LATEX primary commands, on the condition that we know what we are doing. It also allows the use of the `@` sign when naming macros. The `@` sign is used by the inner LATEX constructs (the commands `\makeatletter` and `\makeatother` will then need to appear in the preamble of the main document). This gives us quite some power, but be forewarned that without good working knowledge of the LATEX documentation and its inner workings, we may — by our own efforts — accomplish more damage than good. This is why I recommend only minimal modifications. As our TEX skills reach higher levels, we may try more challenging things ... but for the time being let's stick to the small steps method ...

4 Page layout

Let's begin with the page layout — column size, margins, indents, side notes, headings ... how can we see the invisible? Where are the document elements such as the main column, side notes, captions or footer being placed? For DVI output:

```
\RequirePackage{showframe}
```

or when PDF is the output format:

```
\input{pdf-trans.tex}
\def\boxsh{\boxshow
    {0 .7 .1 RG .2 w}%
    {0 0 .8 RG [2 2]1 d}{}}
\let\shb\boxsh
```

The `showframe.sty` package is a part of a bigger macro package `eso-pic.sty` — it helps to illustrate the physical placement of the page contour, that is, the locations of fixed elements of the column exposition, such as heading, column, footer or side notes (fig. 1).

What are the `\RequirePackage` and all the rest being used for? Well, without delving into technical details of LATEX and the packages being used, more serious tasks can hardly be approached. At first it should suffice to say that we are defining low-level plain TEX macros to suit our own needs. The inquisitive user should consult the package documentation or be happy with a terse: *It is easier that way!* I would like to add that the `pdf-trans` package [3] by Paweł Jackowski, is written in low-level TEX so cleanly that it can be directly incorporated

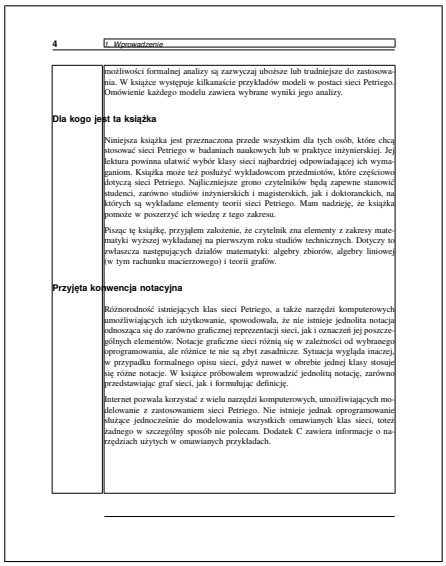


Figure 1: See with your own eyes what the composed page looks like

Figure 2: `\shb` depicts the boxes ...

into LATEX 2_ε. So far, I have not come across any conflict of those macros with any LATEX macros. The `\boxsh` definition, or its abbreviation `\shb` (*showbox*) placed directly before boxes like `\vbox` and `\hbox` will, when typeset, show their contours and baselines (fig. 2). We need only remember to exclude the "helpers" from the final typesetting, e.g.,

```
\let\shb\relax
```

Now we can move on to the next activity, i.e., setting of the page parameters as seen in the printout. The default page sizes defined in LATEX classes usually do not conform to the Polish typography tradition or the requirements of our publishing houses. Nothing stands in the way of adjusting them to our needs. Already in the standard format, there are about 15 parameters which define the characteristic page dimensions, and to which we have direct access. Not everything needs to be "touched" — if we do not use marginal notes in our document, then there is no need to manipulate their dimensions.

Let us for example do the following:

```
\setlength{\paperwidth} {165mm}
\setlength{\paperheight}{238mm}
\special{papersize=\the\paperwidth,
                   \the\paperheight}
```

It doesn't seem like much ... but if at a later stage the document will be further processed at prepress or converted to the PDF format, its pages should have proper outer dimensions (including margins, etc.). Without that magic "spell" the paper size information will be missing from the DVI file and most drivers (e.g., `dvips`) will in such a case assume the default letter page size (11 in × 8.1 in or 279 mm × 216 mm). Those who like minimizing chances for trouble in the future may readily remember and appreciate this little hint.

The modern production cycle of compiling documents directly to PDF from TEX sources, with pdfTEX, also requires these sizes to be set, this time like so:

```
\pdfpagewidth  165mm
\pdfpageheight 238mm
```

Other default size values will surely require corrections. For example:

```
\setlength{\textheight}    {190mm}
\setlength{\textwidth}     {130mm}
\setlength{\hoffset}       {  0mm}
\setlength{\voffset}       {-11mm}
\setlength{\oddsidemargin} { -5mm}
\setlength{\evensidemargin}{ -5mm}

\newlength\g@parindent
\setlength\g@parindent{20bp}

\setlength{\parskip}    {0pt}
\setlength{\parindent}  {\g@parindent}
\setlength{\leftmargini}{\g@parindent}
\setlength{\topskip}    {8bp}
```

The auxiliary variable `\g@parindent` we define above (something like a *global-parindent*) helps other macros to come, where we will define distances being multiples of the paragraph indent. Therefore, it seems sensible to define that particular length unit in one place. This "fixed" variable allows us to not to worry, for example, about some lengths changing with a change of the current font size.

What alternatives exist? One is the `geometry` package — calling it with appropriate parameters allows setting of all necessary dimensions of the typeset page [6]:

```
\RequirePackage[pdftex,
     papersize={185mm,235mm},
     textwidth= 123mm,
    textheight= 181mm,
         inner=  42mm,
    headheight=   9pt,
       headsep=   8mm,
          top=  14mm,
      footskip=  10mm,
marginparwidth=  21mm,
marginparsep=    1mm,
 includehead,
  asymmetric,
]%
{geometry}
```

(Note: the above examples are for two different publications).

We should also mention:

```
\sloppy
\widowpenalty   10000
\clubpenalty    10000
\flushbottom
```

The `\sloppy` parameter forces LATEX to set quite liberal demerit values thus allowing for sub-optimal line breaks and page breaks. The following two parameters, `\widowpenalty` and `\clubpenalty`, set for the maximum demerit value of 10000, forbid page breaks leaving "widows" or "orphans". The `\flushbottom` parameter (a possibly overzealous setting) asks for the following pages to be set to exactly the same height. Some packages may confuse the typesetting with respect to page height.

5 Not only Computer Modern

There is nothing to prevent replacing the "core" TEX fonts with fonts of the typographer's choosing. We will not go here into the problems related to using outline fonts with TEX as they have been discussed many times elsewhere. However, when typesetting math texts, especially with "tough" or "heavy" math, one should bear in mind that only the CM fonts offer the most comprehensive set of sophisticated math signs and symbols.

Recently the free fonts of high TEXnical quality were joined by the TEX Gyre family of fonts. They complement the collection of other freely available computer fonts of Polish origin (Antykwa Półtawskiego, Antykwa Toruńska, Kurier, Iwona, Cyklop) [10].

Lets assume that our model document will use Pagella (previously QuasiPalatino) as the base font (the current, serif font), the sans serif Heros font (in some installations it might lie around as QuasiSwiss) — for typesetting of headings or sometimes to emphasize portions of text, and Cursor (ex-QuasiCourier) as the typewriter font (e.g., for program code snippets). To achieve that one need only give the following commands:

```
\RequirePackage{pxfonts}
\RequirePackage{tgheros}
\RequirePackage{tgpagella}
\RequirePackage[QX]{fontenc}
```

The `pxfonts.sty` macro package enables the use of math symbols available with the Pagella font (the `pxfonts` package in TEX distributions). Inclusion

of `fontenc.sty` with the QX option allows for easy access to the complete set of glyphs available in the font — for more on the (Polish) QX encoding see [9].

All those packages can also be included "wholesale", with the following single command:

```
\RequirePackage{pxfonts,tgheros,tgpagella}
\RequirePackage[QX]{fontenc}
```

6 Typesetting of headings

The fundamental LaTeX commands `\part`, `\chapter`, `\section`, `\subsection`, `\subsubsection`, `\paragraph`, and `\subparagraph` are responsible for the formatting of the titles of the sectioning units of the document. Of course, the commands do much more but for now we are only going to discuss the formatting of their parameter text.

First, to make it easier to set the size of the font and line spacing, let's define some font manipulation utility macros:

```
\def\h@fam{qhv}
\def\font@def #1#2{%
    \usefont{\encodingdefault}%
    {\h@fam}{#1}{#2}}
\def\font@set #1#2#3#4{%
    \font@def{#1}{#2}%
    \fontsize{#3pt}{#4pt}\selectfont}

\def\foo@chp@num{\font@set{bx}{n} {32}{16}}
\def\foo@chp@txt{\font@set{b} {n} {18}{16}}
\def\foo@sec    {\font@set{b} {n} {14}{16}}
\def\foo@sse    {\font@set{m} {n} {12}{14}}
\def\foo@sss    {\font@set{b} {n} {11}{13}}
\def\foo@par    {\font@set{m} {n} {11}{13}}
\def\foo@spa    {\font@set{m} {it}{10}{12}}
```

Why these fancy macro names? Well, indeed, the naming is the writer's choice ... we tried to make them intuitive (font for the headings, chapter text, chapter number and so on) — there is no ideal, unambiguous recipe. Here we only suggest a solution — one can define macros with almost any name. What one should look out for, however, are names previously used inside the format, classes or packages used with our document.

Now we will move on to redefining the original LaTeX macros — the use of `\renewcommand` or `\def` is crucial. The `\def` command is to some degree dangerous: as a purely Plain TeX construct, it does not control name conflicts, thus one may unintentionally create quite some havoc. The `\newcommand` command detects possible name conflicts and so if we really want to redefine an already existing macro, we should use the `\renewcommand` command. Let's start with `\chapter`:

```
\renewcommand\chapter{%
  \if@openright\clearemptydoublepage
```


Przedmowa

Pomysł napisania tej książki zrodził się kilka lat temu, a konkretnie w 2000 roku, kiedy obroniłem pracę doktorską, której tematyka była związana z kolorowanymi sieciami Petriego. Rok wcześniej ukazała się książka [74], której jestem współautorem. Zawierała ona przegląd klas sieci Petriego niskiego poziomu. Brakowało jednak w literaturze książki, w której przedstawiono by wybrane klasy sieci Petriego niskiego i wysokiego poziomu z zachowaniem jednolitych oznaczeń i uwypukleniem różnic między tymi klasami. Postanowiłem tę lukę wypełnić, choć realizacja tych ambitnych planów się odsuwała, głównie z powodu różnych obowiązków naukowych i dydaktycznych. W tym czasie prowadziłem badania dotyczące systemów czasu rzeczywistego. Interesowały mnie możliwości zastosowania sieci Petriego do modelowania i analizy systemów wbudowanych, a zwłaszcza własności czasowych takich systemów.

Udało mi się w końcu doprowadzić do skutku moje zamierzenia. Książka ta nie zawiera przeglądu wszystkich klas sieci Petriego, nie jest też wyczerpującym studium na temat klas w niej opisanych. Zależało mi bowiem na przedstawieniu zarysu teorii sieci Petriego przez ukazanie różnorodności klas tych sieci, metod ich formalnej analizy oraz możliwości ich zastosowania do modelowania systemów współbieżnych i systemów czasu rzeczywistego. Miała to być specyficzna *podróż* przez różne klasy sieci, począwszy od najprostszych, skończywszy na bardzo rozbudowanych z możliwościami hierarchizacji opisu sieci.

Wszystkie omawiane w książce klasy sieci Petriego zostały opisane z podaniem ich formalnej definicji, z przedstawieniem ich najważniejszych własności oraz metod ich analizy. Główny nacisk położyłem na zaprezentowanie przykładów zastosowania poszczególnych klas sieci, zwracając uwagę zwłaszcza na formalną analizę ich własności. W kilku wypadkach ten sam system rozważam przy okazji omawiania różnych klas, tak aby zwrócić uwagę na różne możliwości rozmaitych klas sieci Petriego.

1

Figure 3: The beginning of an unnumbered chapter

```
  \else\clearpage\fi
  \thispagestyle{open}%
  \global\@topnum\z@
  \@afterindentfalse
  \secdef\@chapter\@schapter}
```

— here we call the `\clearemptydoublepage` command with the objective to output, if necessary, an empty left (even numbered) page (with head and foot empty), just before the chapter's title page. This command does not exist in the canonical set of LaTeX macros, so we have to define it ourselves [2]:

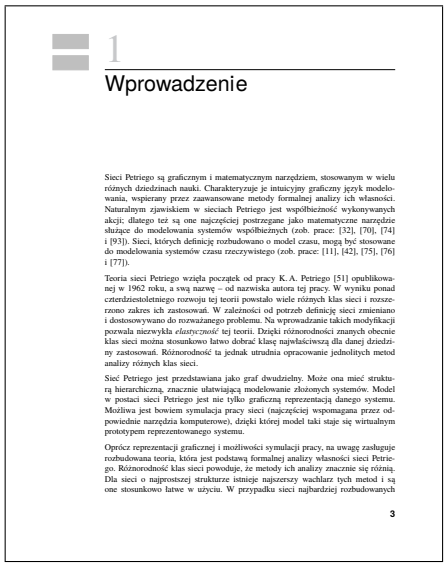
```
\newcommand{\clearemptydoublepage}{%
  \newpage{\thispagestyle{empty}}%
  \cleardoublepage}
```

The use of a nonstandard style for the current page `\thispagestyle{open}` may also have caught the reader's attention — we will discuss this in section 7.

The following changes reach deeper into the source code — they cannot be explained without a detailed analysis of the original LaTeX code — those interested are referred to the documentation. One might also see this as an antidote for sleeplessness. We will cut corners and go directly to the the macro code. The macros modify the headings in the way presented with figures 3 and 4.

Let us begin with some utility definitions (only those which have been used to change the look of the chapters' titles):

```
\definecolor {xxv@gray} {cmyk}{0,0,0,.25}
\newlength\begin@skip
\setlength\begin@skip {46mm}
\newsavebox{\chpt@box}
\def\@stempel{\vphantom
     {\foo@chp@num 0123456789}}
```


1
Wprowadzenie

Sieci Petriego są graficznym i matematycznym narzędziem, stosowanym w wielu różnych dziedzinach nauki. Charakteryzuje je intuicyjny graficzny język modelowania, wspierany przez zaawansowane metody formalnej analizy ich własności. Naturalnym zjawiskiem w sieciach Petriego jest współbieżność wykonywanych akcji; dlatego też są one najczęściej postrzegane jako matematyczne narzędzie służące do modelowania systemów współbieżnych (zob. prace: [32], [70], [74] i [93]). Sieci, których definicję rozbudowano o model czasu, mogą być stosowane do modelowania systemów czasu rzeczywistego (zob. prace: [11], [42], [75], [76] i [77]).

Teoria sieci Petriego wzięła początek od pracy K. A. Petriego [51] opublikowanej w 1962 roku, a swą nazwę – od nazwiska autora tej pracy. W wyniku ponad czterdziestoletniego rozwoju tej teorii powstało wiele różnych klas sieci i rozszerzono zakres ich zastosowań. W zależności od potrzeb definicję sieci zmieniano i dostosowywano do rozważanego problemu. Na wprowadzanie takich modyfikacji pozwala niezwykła *elastyczność* tej teorii. Dzięki różnorodności znanych obecnie klas sieci można stosunkowo łatwo dobrać klasę najwłaściwszą dla danej dziedziny zastosowań. Różnorodność ta jednak utrudnia opracowanie jednolitych metod analizy różnych klas sieci.

Sieć Petriego jest przedstawiana jako graf dwudzielny. Może ona mieć strukturą hierarchiczną, znacznie ułatwiającą modelowanie złożonych systemów. Model w postaci sieci Petriego jest nie tylko graficzną reprezentacją danego systemu. Możliwa jest bowiem symulacja pracy sieci (najczęściej wspomagana przez odpowiednie narzędzia komputerowe), dzięki której model taki staje się wirtualnym prototypem reprezentowanego systemu.

Oprócz reprezentacji graficznej i możliwości symulacji pracy, na uwagę zasługuje rozbudowana teoria, która jest podstawą formalnej analizy własności sieci Petriego. Różnorodność klas sieci powoduje, że metody ich analizy znacznie się różnią. Dla sieci o najprostszej strukturze istnieje najszerszy wachlarz tych metod i są one stosunkowo łatwe w użyciu. W przypadku sieci najbardziej rozbudowanych

3

Figure 4: The beginning of a numbered chapter

```
\def\foo@chp@num{%
    \font@set{qtm}{m}{n}{48}{48}}
\def\foo@chp@txt{%
    \font@set{qhv}{m}{n}{26}{28}}
```

And now to "more serious" code snippets:

```
\newcommand\@make@chapterhead[2]{%
  \nointerlineskip
  \vspace*{-\topskip}%
  \shb\vbox to\begin@skip{%
  \vspace*{-.5mm}%
  \parindent \z@
  \language 255
  \raggedright
  \color{xxv@gray}%
   \ifx|#2|
    \sbox{\chpt@box}%
      {\normalfont\foo@chp@txt
       \textcolor{black}{#1}}%
   \else
    \sbox{\chpt@box}%
      {\normalfont\foo@chp@num#1\@stempel}%
   \fi
  \chap@dinks
  \rlap{\raisebox{10pt}[0pt][0pt]%
        {\usebox{\chpt@box}}}%
  \color{black}%
  \rule{\linewidth}{.5pt}%
  \hfill\endgraf
  \par\vspace{-8pt}%
  \interlinepenalty\@M
  \foo@chp@txt#2\par
  \vspace{\stretch{1}}
  }\nointerlineskip \vskip-6pt}
```

That parameterized macro deals with two cases — numbered and unnumbered chapters. Doing it that way seems to facilitate future changes. Usage of this definition is trivial:

```
% for numbered:
\renewcommand\@makechapterhead[1]{%
  \@make@chapterhead
    {\thechapter}%
    {#1}}%
% for unnumbered:
\renewcommand\@makeschapterhead[1]{%
  \@make@chapterhead
    {#1}%
    {}}
```

Any questions? `\@stempel`? What are the digits 0–9 for? They are not needed in the presented case. The font we use does not require it because all digits have the same height.But let's imagine that the designer employed in that place an unusual, e.g., handwriting font — each digit has its own height and depth ... and the headings must be aligned "in one line". This definition is the easiest way to obtain the maximal dimensions.

Now, let's remind ourselves of how to modify the appearance of the lower level headings — there is little of pure plain but nonetheless:

```
\renewcommand\section{\@startsection
  {section}%            name
  {1}%                  level
  {\z@}%                indent
  {-3.5ex \@plus -1ex \@minus -.2ex}% above skip
  {2.3ex \@plus.2ex}%   below skip
  {\normalfont\foo@sec}% font
  }
```

Similarly for the titles of the lower levels down to `\subparagraph`, if need be. What is left is to insert the period sign after the section number in the numbered titles. The most convenient way is to redefine the proper macro:

```
\def\@seccntformat#1{
  \csname the#1\endcsname.\quad}
```

Low-level, as in ... plain!

7 Changing headers and footers

The default page headers are usually not satisfying. The same applies to footers, where page numbers are usually placed. However, nothing stands in our way of changing that appearance! The manipulations of the content of these page elements can be done according to the following pattern:

```
\def\ps@myheadings{%
  \def\@evenhead{ ...left  head... }%
  \def\@oddhead { ...right head... }%
  \let\@oddfoot { ...left  foot... }%
  \let\@evenfoot{ ...right foot... }%
  \let\@mkboth\markboth
  }
```

Figure 5: Page headers

For example, the headers presented in figure 5 were defined as follows:

```
\newlength\full@wd
\setlength\full@wd{145mm}
\def\foo@hed@pag{%
    \font@set{qhv}{b}{n}{11}{10}}
\def\foo@hed@odd{%
    \font@set{qhv}{m}{n} {9}{10}}
\let\foo@hed@eve = \foo@hed@odd
```

The `\foo@hed@pag` command defines the font to be used when typesetting the page numbers in the header, the `\foo@hed@odd` and `\foo@hed@eve` commands may be used to define different fonts for the left and right running headings. In the sample case illustrated with figure 5, the same font and size were used for both page numbers.

The style definition of the sample page looks as follows:

```
\def\ps@MYheadings{%
  \def\@evenhead{\@x@line
    {\makebox[22mm][l]{\foo@hed@pag\thepage}%
     \foo@hed@eve\leftmark
     \hfill}}%
  \def\@oddhead {\@x@line
    {\hspace*{22mm}\foo@hed@odd\rightmark
     \hfill
     \makebox[22mm][r]{\foo@hed@pag\thepage}}}%
  \let\@oddfoot\@empty
  \let\@evenfoot\@empty
  \let\@mkboth\markboth
}
\def\ps@open{
  \let\@evenhead\@empty
  \let\@oddhead\@empty
  \def\@oddfoot {\makebox[\textwidth][r]{%
        \foo@hed@pag\thepage}}%
  \def\@evenfoot{\makebox[\textwidth][l]{%
        \foo@hed@pag\thepage}}%
}
```

Here we see the previously mentioned, private page style `\thispagestyle{open}`, applied to the initial pages of chapters (figure 3 and 4). In a similar way other needed page styles might be defined.

And now the remaining definitions:

```
\def\@x@line#1{%
    \makebox[\textwidth][r]%
     {\shb\hb@xt@\full@wd{#1}\ul}}
\newcommand\ul{\unskip
  \llap{\rule[-4bp]{\full@wd}{.5bp}}}
```

As we see, LaTeX and plain commands can be used alongside. One only needs to remember their functionality — parametrisation, expandability and redefinitions.

The periods are still missing after chapter and section numbers:

```
\renewcommand\chaptermark[1]{%
  \markboth {\thechapter.\space#1}%
            {\thechapter.\space#1}}

\renewcommand\sectionmark[1]{%
  \markright{\thesection.\space{#1}}}
```

The default page style declaration is simple:

```
\pagestyle{MYheadings}
```

8 Redefining enumeration environments

Our customization zeal will not stop short of standard enumerations: we feel that some things should be done about them.

LaTeX enumeration environments have several variants and applications. However, a set of instructions similar for all of them determines their appearance (hanging indents, line spacing, indents). Unfortunately, we have to consult "the source", retype appropriate parts of the code and modify the relevant parameters. Let's do it:

```
\setlength\leftmargini  {\g@parindent}
\leftmargin\leftmargini
\setlength\leftmarginii {\g@parindent}
\setlength\leftmarginiii{\g@parindent}
\setlength\leftmarginiv {\g@parindent}
\setlength\leftmarginv  {1em}
\setlength\leftmarginvi {1em}
\setlength\labelsep     {.5em}
\setlength\labelwidth   {\leftmargini}
\addtolength\labelwidth {-\labelsep}
\setlength\partopsep{0\p@}
```

`\g@parindent`, defined close to the beginning of our code, has now found its application — now, independent of their placement within the document, enumerations will always be indented by the same amount. Unless we spoil this with a strange macro or environment.

In our next move we will change the formatting parameters for the consecutive enumeration nesting levels of which standard LaTeX allows up to six! Such deep nesting is not used in practice. One should think of re-writing the text rather than using very deeply nested enumerations, which might hinder comprehension. Anyway, let's now deal with the definitions:

```
\def\@listi{\leftmargin\leftmargini
  \parsep  \z@
  \topsep  .5\baselineskip
     plus .25\baselineskip
    minus .15\baselineskip
  \itemsep \z@ plus .5pt}
\let\@listI\@listi
\@listi
```

And the deeper levels:

```
\def\@listii {\leftmargin\leftmarginii
    \labelwidth\leftmarginii
    \advance\labelwidth-\labelsep
    \topsep  \z@
    \parsep  \z@
    \itemsep \z@}
\def\@listiii{\leftmargin\leftmarginiii
    \labelwidth\leftmarginiii
    \advance\labelwidth-\labelsep
    \itemsep   \z@}
```

. . . enough! Those who grasped the rules of the game will easily tackle even deeper levels if need be. and, as demonstrated, that kind of activity requires reading the source code of classes and styles.

Let's change the bullets for the consecutive nesting levels (we will skip the last two levels):

```
\renewcommand\labelitemi  {\sq@black}
\renewcommand\labelitemii {\sq@white}
\renewcommand\labelitemiii{\textemdash}
\renewcommand\labelitemiv {$\ast$}

\def\sq@black{\rule[.1ex]{1ex}{1ex}}
\def\sq@white{\mbox{%
    \fboxsep=0pt
    \fboxrule=.5pt
    \raisebox{.15ex}{\fbox{%
      \phantom{\rule{.8ex}{.8ex}}}}}}
```

The following command might turn out to be very useful:

```
\def\keepitem{\@beginparpenalty\@M}
```

— we forbid page breaks before the first short item. It suffices to give it just before that item — this is the equivalent of eliminating "orphans" in running text.

For those liking pure LaTeX solutions, the package `enumitem` is worth recommending. It allows for individual tailoring of each enumeration through proper parameters, or might be applied globally, in the preamble:

```
\setenumerate{%
    labelsep = 6pt,
  leftmargin = \leftmargini,
     itemsep = 1pt,
      topsep = 6pt,
   partopsep = 0pt,
      parsep = 0pt
 }
\setitemize{%
       label = \textsquare\hfill,
    labelsep = 6pt,
  leftmargin = *,
     itemsep = 1pt,
      topsep = 6pt,
   partopsep = 0pt,
      parsep = 0pt
  }
```

The meanings of the parameters are intuitive. Their names correspond to the various lengths used to construct enumerations. A more complete description of the package might be found in its documentation [1].

9 Modifying footnotes

Another typesetting element we might want to modify is footnotes. Let's assume we'd like the rule separating the footnote from the text column to be of full column width:

```
\renewcommand\footnoterule{%
  \kern-3\p@
  \hrule width\linewidth height.5pt
  \kern2.5\p@}
```

That's the way to gain direct access to the separating rule: we can modify its length, thickness, position and even colour!

With the replacement of the default font, need may arise to correct the positioning of footnote numbers both in the running text and the footnotes. Changes might be needed in the font and size of the indexes, or the footnote marks.

```
\def\@makefnmark{\hbox{%
 \@textsuperscript{\normalfont\@thefnmark}}}

\renewcommand\@makefntext[1]{%
 \parindent\g@parindent%
 \noindent
 \hb@xt@\parindent{\hss\@makefnmark\space}#1}
```

10 Positioning floating objects

Figures and tables are amongst the most often used floating objects. One may also encounter other objects with distinct typographic properties (algorithms, screen shots) and separate numbering. The placement of such typesetting elements is controlled in LaTeX through several parameters which, unfortunately, have their defaults set to rather restrictive

values. For publications with a large number of tables and figures, it might be difficult to obtain aesthetically pleasing page breaks and proper placements of the elements with respect to the running text.

However, again nothing stands in our way to slightly relax those parameters. For example:

```
\setcounter  {topnumber}            {3}  %%{2}
\renewcommand{\topfraction}         {.9} %%{.7}
\setcounter  {bottomnumber}         {0}  %%{1}
\renewcommand{\bottomfraction}      {.2} %%{.3}
\setcounter  {totalnumber}          {3}  %%{3}
\renewcommand{\textfraction}        {.1} %%{.2}
\renewcommand{\floatpagefraction}   {.85}%%{.5}
\setcounter  {dbltopnumber}         {2}  %%{2}
\renewcommand{\dbltopfraction}      {.8} %%{.7}
\renewcommand{\dblfloatpagefraction}{.8} %%{.5}
```

The meaning of the particular counters and parameters can be found in the LaTeX documentation. The default values offered by the format and standard LaTeX classes are given after the double percent sign. The modified values will surely cause a better placement of our floating objects.

11 Smaller fonts for tables et al.

Instead of placing a command changing the current font size in each and every `table` environment, we may achieve the desired result globally:

```
\def\font@caption{\font@set{qtm}{m}{n}{8}{10}}
\def\@floatboxreset {%
    \reset@font
    \font@caption
    \centering
    \@setminipage
 }
```

Isn't it simple? In a similar way the e.g., `verbatim` environment might be modified — a slight decrease of the typewriter font size without reducing the readability of these fragments. This may be achieved with a small change to the canonical LaTeX format:

```
\def\verbatim@font{%
    \normalfont\ttfamily\small}
```

However, one should bear in mind that some specialized packages dealing with verbatim-like environments will be immune to such machinations. Such packages might have internal font mechanisms and offer considerable flexibility.

12 Summary

The topic has by no means been exhausted — but still, I hope that the examples given here will convince those less versed in LaTeX that even that format may be modified to achieve one's own ends.

Turning such code snippets into styles or classes is a separate subject and, often, requires more in-depth studies of the subject, e.g., compatibility with other macro packages. There will be no way around reading the documentation or, in many cases, analysing the TeX code.

Some examples in this article were taken from this book by Marcin Szpyrka: *Sieci Petriego w modelowaniu i analizie systemów współbieżnych*, Warsaw 2007. The book was typeset for Wydawnictwa Naukowo-Techniczne using the techniques presented here.

References

[1] Javier Bezos: *Customizing lists with the enumitem package*, `mirror.ctan.org/macros/latex/contrib/enumitem/`

[2] Michel Goossens et al.: *The LaTeX Companion*, Addison-Wesley, 2nd edition (2004)

[3] Paweł Jackowski, *Box Transformations in pdf-trans.tex*, `mirror.ctan.org/macros/generic/pdf-trans/`

[4] Helmut Kopka, Patrick W. Daly: *Guide to LaTeX 2ε, Document Preparation for Beginners and Advanced Users*, Addison-Wesley, 4th edition (2003)

[5] Leslie Lamport: *LaTeX System opracowywania dokumentów Podręcznik i przewodnik użytkownika* [LaTeX: A Document Preparation System], Wydawnictwa Naukowo-Techniczne (2004)

[6] Hideo Umeki: *The geometry package*, `mirror.ctan.org/macros/latex/contrib/geometry/`

[7] Marcin Woliński: *Moje własne klasy dokumentów* [My own document classes], `www.math.upenn.edu/tex_docs/latex/mwcls/mwclsdoc.pdf`

[8] Marcin Woliński: *Ku polskim klasom dokumentów dla LaTeX-a*, Biuletyn GUST, nr 15, 2000

[9] *Kodowanie QX* [The QX encoding], `www.gust.org.pl/doc/fonts/qx/`

[10] *Projekty fontowe* [Font projects], `www.gust.org.pl/doc/fonts/projects`

◇ Jacek Kmiecik
AGH University of Science and Technology
University Computer Centre
al. Mickiewicza 30, 30-059 Kraków
`jk (at) agh dot edu dot pl`