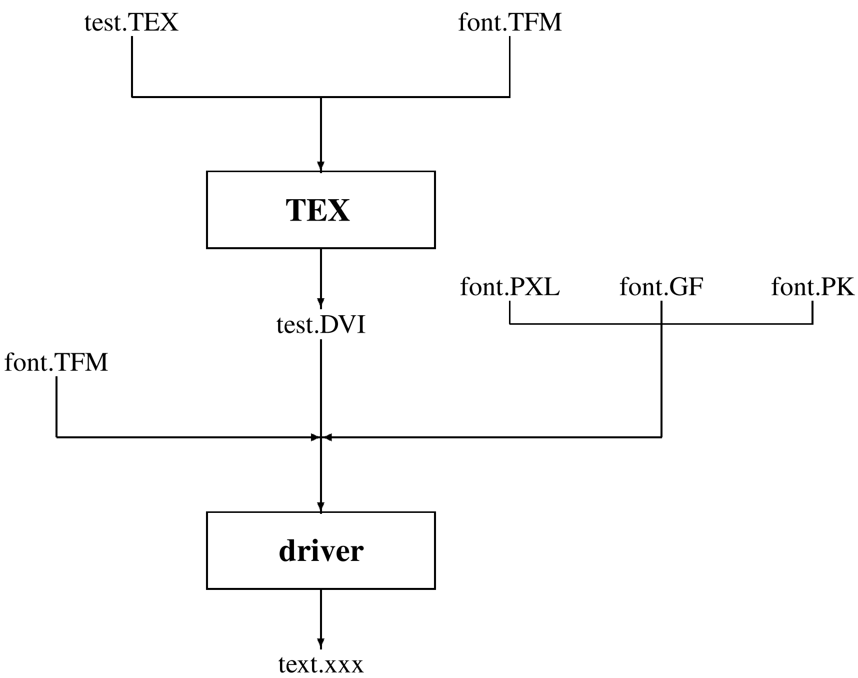
BIJLAGE Z

TEX structuurschema's

Huub Mulders

1 De structuur van TEX

Hieronder volgt een schema waarin is aangegeven welke bestanden er worden gebruikt en aangemaakt wanneer met behulp van TEX van een tekstbestand een *gezette* tekst wordt gemaakt.



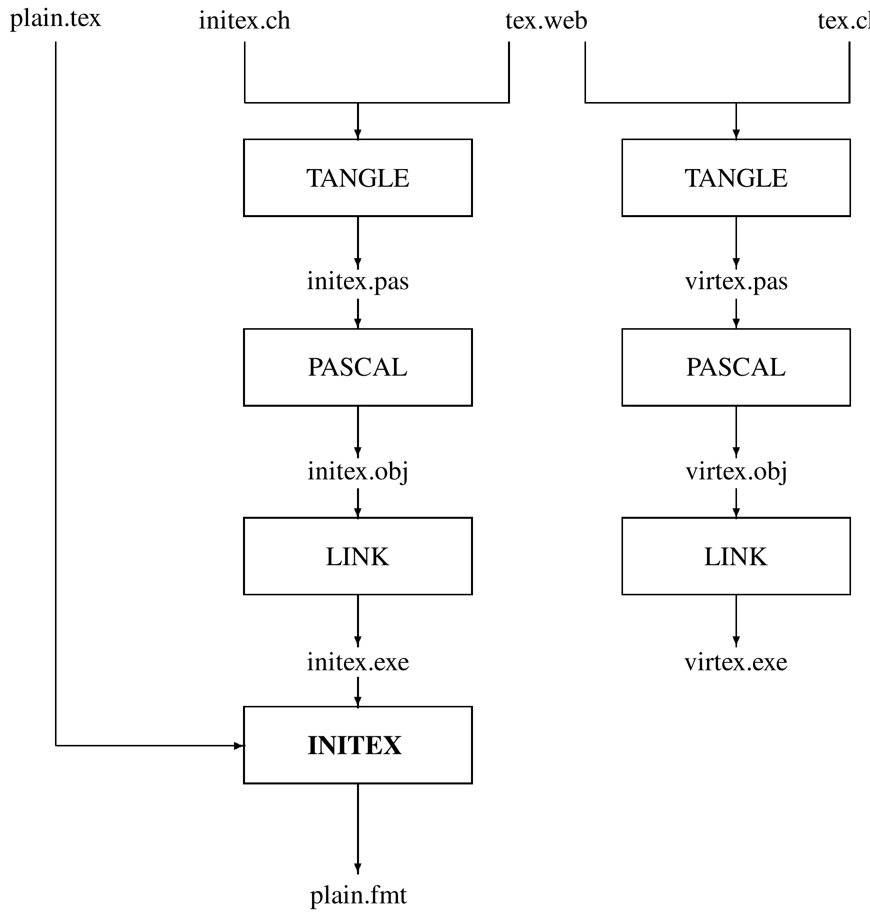
2 Overzicht filetypes die bij TEX etc. een rol spelen

Hieronder volgt een overzicht van de verschillende filetypes, aangegeven door een *suffix*, die bij TEX een rol spelen. Merk op dat sommige besturingssystemen, zoals MS/PC-DOS, de *suffixes* beperken tot drie symbolen.

Specifiek voor TEX etc.		
.aux	:	TEX hulpfile; gemaakt en gebruikt door TEX (en o.a. LATEX)
.base	:	METAFONT format-file (geprecompileerde macro's, "plain"); gemaakt door INIMF, gebruikt door MF of gepreload door VIRMF
.bbl	:	BIBTEX outputfile; gebruikt door LATEX
.bib	:	BIBTEX inputfile
.bst	:	BIBTEX documentstyle
.ch	:	aanpassingen van .web-files; gebruikt door TANGLE en WEAVE
.doc	:	1. documentatie
	:	2. LATEX documentstyle, met kommentaar (verder gelijk aan .sty)
.dvi	:	device independant document; gemaakt door TEX; gebruikt door drivers
.fmt	:	TEX format-file (geprecompileerde macro's, "plain") gemaakt door INITEX, gebruikt door TEX of gepreload door VIRTEX
.*gf	:	generic font: pixels (*=300xvergroting); gemaakt door METAFONT, gebruikt door drivers
.glo	:	LATEX glossary (hulpfile)
.idx	:	LATEX index (hulpfile)
.lis	:	o.a. TEX/METAFONT logfile
.lof	:	LATEX list-of-figures (hulpfile)
.lot	:	LATEX list-of-tables (hulpfile)
.mf	:	METAFONT inputfile
.*pk	:	packed font pixels (*=300xvergroting); gemaakt uit .*gf door GFTOPK, gebruikt door drivers
.pool	:	stringpool; gemaakt door TANGLE, gebruikt door bijbehorende .exe
.*pxl	:	font pixels (*=1500xvergroting); gemaakt uit .*gf door GFTOPXL, gebruikt door drivers
.sty	:	LATEX documentstyle of style-optie
.tex	:	TEX/LATEX/SLITEX inputfile
.toc	:	TEX table-of-contents (hulpfile)
.typ	:	"leesbaar" gemaakte .dvi; gemaakt door DVITYPE
.tfm	:	TEX font metrics; gemaakt door METAFONT, gebruikt door TEX drivers
.web	:	sources van diverse programma's; gebruikt door TANGLE en WEAVE

3 Het genereren van een nieuwe TEX versie

Hieronder volgt een schema waarin staat aangegeven hoe een nieuwe versie van de TEX programmatuur kan worden gemaakt. Hier wordt uitgegaan van PASCAL als programmeertaal. De mogelijkheid bestaat ook om in een andere programmeertaal te werken, zoals C. In dat geval is er een speciale versie van WEB nodig.



Het maken van een `TeX.exe` en `LaTeX.exe` is o.a. afhankelijk van de gebruikte TeX.ch en/of van welke 'firma' de TEX vandaan komt.

- Voor de TeX van de DECUS-tape:

- Voor Kellerman & Smith moeten bij het linken van initex.obj en virtex.obj een aantal svi*.obj files mee-gelinkt worden. Dan kan `(La)TeX.exe` gemaakt worden zoals bij DECUS-tape.
- Meestal is er wel een documentatie file waarin staat hoe een `TeX.exe` te maken
- TEX en LATEX kunnen altijd gedraaid worden via het commando:

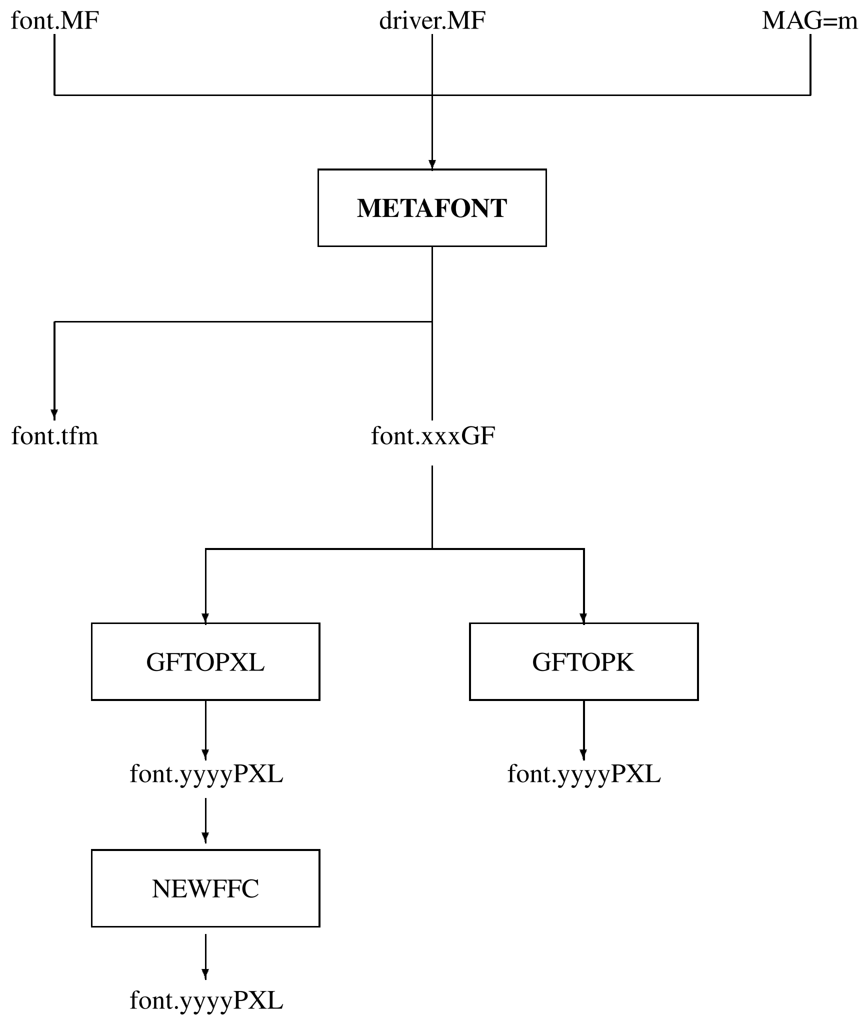
  ```
  VIRTEX &plain filename
  ```

  of:

  ```
  VIRTEX &lplain filename
  ```

4 De structuur van METAFONT

Hieronder staat schematisch aangegeven welke bestanden er worden gebruikt en aangemaakt wanneer met METAFONT *fonts* worden gemaakt. In het schema zijn ook diverse nabehandelingsprogramma's (`..TO..`) aangegeven:



Vergrotingen en naamgeving van fontfiles

magstep		0	0.5	1	2	3	4	5	6	7	8	9
mag*1000	(m)	1000	1095	1200	1440	1728	2074	2488	2986	3583	4300	5160
gf	(xxx)	300	329	360	432	518	622	746	896	1075	1290	1548
pxl	(yyyy)	1500	1643	1800	2160	2592	3110	3732	4479	5375	6450	7740

BIJLAGE AA

The 1990 DECUS TEX Collection

Announcement
Ted Nieland

(news from TeXHaX issue 62)
4 september 1990

The DECUS Languages and Tools SIG Public Domain Working Group and the Electronic Publishing SIG TEX/LATEX/WEB Working Group are proud to announce the 1990 DECUS TEX Collection. This collection offers nearly everything a TEX User would want on their system for TEX.

The master tapes for the collection have been sent to DECUS Library and to the top of the DECUS LUG distribution tree. The new collection will be available to all shortly through their channel for procuring DECUS Software.

This collection is an extensive rework of the previous collection with nearly all of the material being updated or new. More DVI drivers have been added and many of the VMS programs now sport a CLD interface.

Also, an extensive effort on documentation has taken place resulting in a DECUS TEX Help Library.

The following items are included in the DECUS TEX Collection 1990:

- WEB (Tangle 4.0 / Weave 4.1)
- TEX Version 3.0
- METAFONT Version 2.0
- LATEX Macro Package 2.09 <7 Dec 1989> (with mod for TEX 3.0)
- SliTEX Macro Package 2.09 <7 Dec 1989> (with mod for TEX 3.0)
- BibTEX Version 0.99c
- TEXsis Macro Package Version 2.13
- DVIOUT Version 1.2
- DVIPS for VMS, Version 5.35
- DVItoVDU Version 3.2
- DVItoLN03 Version 3.1-4
- XDVI (with support for DecWindows)
- TEXx (with support for DecWindows)
- Vassar Spell Version 2.2
- FWEB (including support for VMs)
- CWEB (including support for VMS)
- MWEB
- TIB
- CRUDETYPE
- DVIDIS (for VAXStations Running VWS)
- GPLOT 4.23
- RNOTOTEX
- IDXTEX
- GloTEX
- DVIDVI
- MAKEINDEX
- PicTeX
- TEXTYL
- DVI2TTY
- LSE Templates for LATEX and BibTEX
- MFWARE (GFtoPK, GFtoPX, etc)
- PICMODE
- TR2TEX
- WS2LATEX
- AmS-TEX Macro Package
- AmS-LATEX Macro Package
- PHYZXX Macro Package
- PHYSE Macro Package
- ScriptTeX Macro Package
- MuTEX Package (including METAFONT files)
- Clarkson LATEX & BibTEX Style Collections
- DECUS TeX Help Library
- Beebe Utah DVI Driver Collection with additional submissions
- DVI2PS
- Many Font Additions (Concrete, Duerer, Chess, DECUSLOGO, among others)
- Support for Foreign Languages including Dutch, French, German, Greek, Hebrew, Icelandic, Italian, Japanese, Korean, Portuguese, Russian, Spanish, Thai, Turkish, and Vietnamese.
- TEX for the Amiga with some DVI Drivers and the LATEX Picture Editor (LPE)
- TEX for the Macintosh (OzTEX), along with BibTEX, and DVI drivers.
- TEX for MS-DOS, plus previewers and DVI drivers.
- Various TEXware for UNIX, including WEB2C and XTEX (for DECStations).

The following output devices are supported:

- DEC LN03 (requires a RAM Cartridge) [DVITOLN03]
- DEC LN03 Plus (uses bitmaps) [DVIL3P]
- DEC LA75 [DVI175]
- PostScript (LPS40, Apple LaserWriter, LN03S) [DVIALW, DVIPS, DVIOUT, GTEX]
- Hewlett Packard Laserjet [DVIJET]
- Hewlett Packard Laserjet Plus [DVIJEP]

- Cannon Engine Laserprinter [DVICAN]
- EPSON Printer [DVIEPS]
- Printronix Printer [DVIPRX]
- Okidata Pacemark 2410 (72 or 144 DPI) [DVIOKI]
- VT terminals, ReGIS Terminals, Tektronix Terminals [DVITOVDU]
- VAXStations running VWS [DVIDIS]
- DECWindows [XDVI, TEXX]
- Version 3.10 BBN BitGraph Terminal [DVIBIT]
- Golden Dawn Golden Laser 100 printer [DVIGD]
- Imagen imPRESS-language laser printer family [DVIIMP]
- Apple Imagewriter 72 or 144 dpi printers [DVIM72 or DVIMAC]
- MPI Sprinter 72 dpi printer [DVIMPI]
- Toshiba P-1351 180 dpi printer [DVITOS]
- Generic Output [DVI2TTY]
- QMS Laser Printers [GTEX]

The collection includes numerous example files including A Gentle Introduction to TeX by Micheal Doob and Essential LaTeX by Jon Warbrick.

For more information on getting a copy of the DECUS TeX Collection, contact your DECUS Local User Group or the DECUS Library at:

DECUS Library (BP02)
219 Boston Post Road
Marlboro, MA 01752-1850
(508) 480 3418/3659/3446

BIJLAGE BB

New Books on TEX

Victor Eijkhout
University of Illinois at Urbana-Champaign
305 Talbot Lab
104 S. Wright Street
Urbana, Illinois 61801, USA
u641001@HNYKUN11

There is a piece of good news to be reported: two new books on TEX have appeared recently, one for beginning to intermediate users, and one for intermediate to advanced users. And there's more good news: TEX is so widely spread that both books originated in Germany, and are written in German. Of the introductory book translations into English and Dutch exist, but the advanced book, in more than one respect the more interesting of the two, has not been translated yet.

TEX books in other languages than English are a good thing for two reasons. One is that they give an indication of the widespread use of TEX. The other is that, quote Norbert Schwartz, author of 'Einführung in TEX' [1], such books are 'a bit more internationally oriented than a book of English or American origin would probably be'. This is especially apparent in 'TEXfür Fortgeschrittene' [2] by Wolgang Appelt, which has a whole chapter on 'Deutschsprachige Text', containing sensible remarks that are relevant to more languages than just German – although they are not particularly relevant to the somewhat plebeian English language.

Introduction to TEX

'Introduction to TEX' by Norbert Schwartz assumes no knowledge of TEX whatsoever, indeed the first chapter 'General information' gives a short list of the merits of TEX. This makes for a nice and motivating introduction for the complete novice.

The same holds for chapter 2, 'Operation', that contains, after a few pages of braces, backslashes, and punctuation, a first example of the use of TEX. Some thirty commands are used here. Obviously the author wants to get the reader going: the details will come later.

Chapter 3 was written in the same vein. In 30 pages a large amount of information about 'Setting text' is given to the reader, with lots of examples. However, this chapter had me frowning a number of times. It is the author's style of writing to use bursts of creativity like

```
\obeylines\everypar{\hfil}
```

to introduce the concepts of `\everypar` and `\parfillskip`, but it wouldn't be mine. And I object to

```
\centerline{\it The current page has
    the number \folio}
```

Fortunately, some important concepts are explained more fully in chapters on macros and 'How TEX works' – although I feel that the section on modes is a bit skimpy. There are two nice chapters on mathematical typesetting, there is a short chapter on output routines, and I was particularly pleased with the chapter on 'Tables and alignment'.

Main part of the appendix to this book is an 80 page (!) list of all TEX and plain TEX commands. The explanations are short, but certainly not cryptic, and often an illuminating example is given. Definitely a good idea of the author.

TEX for the advanced

'TEX for the advanced' by Wolfgang Appelt is a very different book. The subtitle, 'Programming techniques and macro packages' is probably the best indication to its contents. Wolfgang Appelt argues in the preface the need for high level macro packages, and then sets out to assist the reader in constructing such packages. He does this in three ways.

The preface, the introductory chapter, and a chapter 'Macro packages' give general thoughts on how macro packages should be structured, and what their nature should be. He distinguishes between the logical structure and the layout structure, and for both of these the generic and the specific structure. It is useful to have such concepts explained in some detail, and the reader won't hear me arguing the author's point of view.

Pure TEX theory is treated on chapters on 'Spaces' – such a chapter must be answering many prayers of desperate TEXers – and 'Macros and parameters'. The author has a very clear style of explaining, but his explanation of conditionals, sufficient for most cases, distorts the truth a bit.

Lastly, four chapters can be classified as 'case studies in macro package design'. They treat the subjects of a font selection scheme, text structures (lists and sectioning), referencing (including table of contents), and adaptations necessary for the German language. These chapters give complete sets of macros, and they are well explained.

Appelt makes no attempt at being complete. Mathematical typesetting and alignments are not treated in this book, and output routines are hardly touched upon. Given the size of the book this would not have been possible, and concentrating on a few selected topics is probably a good idea.

In all, this book is maybe not sufficient reason to start learning German – which means you'll never make such delightful discoveries as that 'ragged right' is 'Flattersatz', 'flutter setting' in German – but if you know a smattering of the language it certainly won't harm you to pick up this book.

How does it look?

When a book about TEX appears, there is an obvious question: 'has it been done in TEX?' For both books reviewed here the answer is yes, but the results are widely different. The Appelt book is set in 12 point Computer Modern with non-obtrusive headings, which gives a surprisingly open and readable page. Of the Schwartz book I have only seen the Dutch and English translations, which are totally unlike each other. The English translation is set in Computer Modern, photographically reduced to 10 point. Unfortunately, the book was printed rather lightly, which makes the page appear somewhat vague.

The Dutch branch of Addison-Wesley must have been in an adventurous mood, combining New Century Schoolbook as a text face with Avant Garde headings. Choosing Courier as the type writer font was not the optimal choice, but the overall result is rather pleasant – even though there have been a few accidents in typesetting the examples.

As a conclusion I would state that both books are an asset to the TEX community. Neither book is a definite TEX bible, but for both there is certainly a niche to fill.

[1] **Einführung in TEX**, Norbert Schwartz, Addison-Wesley Verlag, Bonn 1988(?) ISBN 3-925118-97-7
Inleiding TEX, Norbert Schwartz, Addison-Wesley Europe, Amsterdam 1990, ISBN 90-6789-151-7
Introduction to TEX, Jost Krieger and Norbert Schwartz, Addison-Wesley Europe, Amsterdam 1989, ISBN 0-201-51141-X
[2] **TEX für Fortgeschrittene**, Wolfgang Appelt, Addison-Wesley Verlag, Bonn, 1988, ISBN 3-89319-115-1

TEX for the impatient

One of the aspects of TEX that set it apart from other text processors is the fact that there exists an ultimate reference: the TEX book. Like its introduction states, this book is for people who have never used it before and the experienced hackers alike. 'TEX for the impatient' makes a similar claim: Paul Abrahams, the senior author, asked himself 'What kind of book would have made it easier for me to learn TEX? What kind of book would I need now, as a more experienced user, to locate commands or functions that I never learned or only half remember?'. In my opinion he, and co-authors Karl Berry and Kathryn Hargreaves, have given an succesful answer to the first question. My thoughts on the second question follow below.

'TEX for the impatient' has a very appealing front cover: the white rabbit from 'Alice in Wonderland' (the one that exclaims 'Oh dear, I shall be too late!') is sitting, looking at his watch, very impatiently. The inside of the book looks good. Computer Modern is used for the text, with a surprising but very satisfactory choice of Optima bold for headings and command names when these are used as headings. Thirteen chapters and an index make up the approximately 360 pages of the book.

After two inevitable chapters 'Using this book' and 'Using TEX', follows an interesting third chapter: 'Examples'. Ten page-long examples with the input on the facing page give a good impression of TEX's capabilities, and give the novice a source of commands and constructs to study (and copy).

Chapter four 'Concepts' starts the reference part of the book. Instead of merging the list of concepts treated here into the table of contents, the authors decided to print it separately on the inside of the back cover. An unusual idea, but I like it. The list is some 90 terms long, and the chapter spans 55 pages. Individual concepts are therefore treated briefly but the explanations are clear and well-written, and there are many references to the subsequent chapters which treat individual commands. In this chapter I appreciated especially the fact that the authors use the anatomical analogy for TEX's workings, and refer to it repeatedly.

Although the authors suggest that novices, after having read chapters 1–3, start working from the summary of commands (chapter 13), looking up commands and concepts, I feel that chapter 4 is really also part of the introduction to TEX. Call it a higher introduction.

The following chapters 5–9 treat TEX commands, grouped by subject. Here too the explanations are clear, but they are less complete than I would like them. It has been a wise decision not to treat each command separately, but to tackle a few commands at a time, for instance `\hss` and `\vss`, or `\unskip`, `\unkern`, and `\unpenalty`.

Chapters 10–12 are probably a good selling point for this book: let it suffice that the titles are 'Tips and techniques', 'Making sense of error messages', and 'A compendium of useful macros'. This last chapter contains an 'extended plain format', containing such much wanted macros as for cross references and left aligned display equations. Explanations of these macros limit themselves to explanations of the way to use them. A 'Capsule summary of commands' and an index complete the book.

On the whole, I find this book very clearly written, and all its information is readily accessible. However, I was a bit annoyed by the small errors that I found. For instance, the delimiters around `\..withdelims` commands don't grow as the authors claim; they are determined by font parameters 20 and 21 of the symbol font. Also, the remarks about the depth (height) of a `\vbox` (`\vtop`) on pages 52 and 161/2 are at odds; in principle this dimensions is the depth (height) of the last (first) box or rule. On page 52 it is stated that theses dimensions are zero if the last (first) item is kern or glue – which is wrong for the `\vtop`; on page 161/2 it is stated that it is zero if the last (first) item is not a box or rule – which is wrong for the `\vbox`. For the exceptions to these statements consider whatsits.

As I mentioned above, the question underlying this second part of the book started 'What kind of book would I need now, as a more experienced user'. By 'experienced user' the authors apparently do not mean an aspiring TEX hacker, since this book explains commands as such, but much less the large scale mechanisms connecting them.

For instance, one technique in chapter 10, 'Leaving space at the top of page', is treated in a mere five lines: the reader is told that `\vskip` does not work, but that `\topglue` does. I was particularly struck by this, since I didn't know the latter command, which is a late addition to TEX version 3. Neither here, nor in the systematic reference chapters is it mentioned whether this is a macro or a primitive. That information can only be found in the command summary; it is not even in the index, like it is in the TEX book.

Another example: page 86 states that 'When TEX breaks a page, it discards any sequence of glue, kerns, and penalty items that follows the break'. This is rather a simplification of what really happens; one might even say that this is simply not true. However, it is a convenient way of looking at things, and as long as you stick to the plain TEX output routine you never notice the difference.

The most obvious sign that the authors do not aim at TEX hackers is of course the fact that they repeatedly refer to the TEX book for the details. On page 167 it even says 'if you want to get adventurous you can learn all about it from pages [. . .] of the TEX book'.

In general, this book gives good factual information, and the information is very easy to find. What it lacks are the explanations, not of commands but of mechanisms. For instance, expansion is never treated as such. Also the problems of expansion and timing in macros for cross references and table of contents are not discussed.

But most people will not write such macros, and since some very handy macros are given in chapter 12, this book can be useful for people wanting to understand and modify or extend existing macros. And as an introduction, it is simply a good book.

BIJLAGE CC

An indentation scheme

Victor Eijkhout
University of Illinois at Urbana-Champaign
305 Talbot Lab
104 S. Wright Street
Urbana, Illinois 61801, USA
u641001@HNYKUN11

Indentation is one of the simpler things in TEX: if you leave one input line open you get a new paragraph, and it is indented unless you say `\noindent`. And if you get tired of writing `\noindent` all of the time, you declare

```
\parindent=0pt
```

at the start of your document. Easy.

More sophisticated approaches to indentation are possible, however. In this article I will sketch a quite general approach that can easily be incorporated in existing macro packages. For a better appreciation of what goes on, I will start with a tutorial section on what happens when TEX starts a paragraph.

1 Tutorial: paragraph start

When TEX is not busy typesetting mathematics, it is processing in *horizontal mode*, or *vertical mode*. In horizontal mode it is putting objects – usually characters – next to each other, in vertical mode it is putting objects – usually lines of text – on top of each other.

To see that there is a difference, run the following pieces of code through TEX:

```
\hbox{a}
\hbox{b}
\bye
```

and

```
a
\hbox{b}
\hbox{c}
\bye
```

You notice that the same objects are treated in two different ways. The reason for this is that TEX starts each job in vertical mode, that is, stacking material. In the second piece of input TEX saw the character 'a' before it saw the boxes. A character is for TEX the sign to switch to horizontal mode, that is, lining up material, and start building a paragraph.

Commands that can make TEX switch to horizontal mode are called 'horizontal commands'. As appeared from the above two examples characters are horizontal commands, but boxes are not. Let us now look at the two most obvious horizontal commands: `\indent` and `\noindent`.

1.1 `\indent` and `\noindent`

`\indent` is the command to start a paragraph with indentation. TEX realizes the indentation by inserting a box of width `\parindent`. If you say `\indent` somewhere in the middle of a paragraph you get some white space there, caused by the empty box.

`\noindent` is the command to start a paragraph without indentation. After this command TEX merely switches to horizontal mode; no indentation box is inserted. If you give this command somewhere in the middle of a paragraph it has no effect at all.

If TEX sees a horizontal command that is not `\indent` or `\noindent`, for instance a character, it acts as if the command was preceded by `\indent`. This is why paragraphs usually start with an indentation.

As an illustration here is a small variation on the above two examples:

```
\noindent
\hbox{a}
\hbox{b}

\indent
\hbox{a}
\hbox{b}
\bye
```

1.2 `\everypar`

TEX performs another action when it starts a paragraph: it inserts whatever is currently the contents of the token list `\everypar`. Usually you don't notice this, because the token list is empty in plain TEX (the TEX book [3] gives only a simple example, and the exhortation 'if you let your imagination run you will think of better applications'). LATEX [5], however, makes regular use of `\everypar`. Some mega-trickery with `\everypar` can be found in [2].

• Just to show how this works, I put in front of this paragraph the statement

```
\everypar={$\bullet\quad$}
```

That is, I told TEX that `$\bullet\quad$` should be inserted in front of a paragraph.

• There's nothing specified for this paragraph; I get the bullet for free, as `\everypar` does exactly what its name promises: it is inserted in front of *every* paragraph.

At the end of the previous paragraph I specified

```
\everypar={}
```

so nothing is inserted from this paragraph onwards.

1.3 Removing indentation

Every TEX user knows that indentation can be prevented globally by setting `\parindent` to zero. However, this is rather crude, and if you use the plain TEX macros you may notice several rather unpleasant side effects of this action, for instance when you use the macros `\item` and `\footnote`.

It is possible to use `\everypar` to prevent indentation, or more correctly: to remove indentation. This can be achieved by

```
\everypar={{\setbox0=\lastbox}}
```

This needs some explanation.

If the last item that was processed by TEX is a box, then that box is accessible by the command `\lastbox`. If the last item was not a box then `\lastbox` is an empty box, but no error ensues. As the `\everypar` list is inserted after any indentation box, the `\lastbox` command will get hold of the indentation box if there is one. By assigning the last box to another box register – here `\box0` – it is removed from where it was previously.

Finally, the statement

```
\setbox0=\lastbox\
```

is enclosed in braces. TEX's grouping mechanism restores values when the group ends that were current when the group began. In this case it has the effect of totally removing the indentation box: first it is taken and assigned to `\box0`, then the value of `\box0` is restored to whatever it was before the group began.

1.4 Other actions at the start of a paragraph

In the above discussion I have omitted one action that takes place at the start of a paragraph: TEX inserts (vertical) `\parskip` glue above the paragraph. As this has no relevance for the subject of indentation I will not go into it any further. However, in a subsequent article I will give more information about `\parskip`.

2 To indent or not to indent

In classical book typography [4] every paragraph is indented, with the exception of the first paragraph of a chapter. Nowadays a design where no paragraph indents is quite common. There are two mixtures between always indenting and never indenting: occasionally indenting, and occasionally not indenting. Thus it seems possible to characterize indentation strategies by two yes/no parameters: one that decides whether paragraphs should indent in principle, and another parameter that can overrule those decisions. Let us now see how this can be implemented in TEX.

2.1 Implementation

Above I have already indicated that changes to `\parindent` should be avoided. Let us then assume that `\parindent` is greater than zero, even if we will never indent a paragraph (see [1] for other uses for the `\parindent` quantity). We must then realize unindented paragraphs by removing their indentation as explained above.

First we need a macro for removing the indentation:

```
\def\removeindentation
    {{\setbox0=\lastbox}}
```

Then we need the switches that control indentation:

```
\newif\ifNeedIndent %as a rule
\newif\ifneedindent %special cases
```

Now for the definition of `\everypar`. This is a bit tricky.

Let us first collect some bits and pieces. The main question is to decide when `\removeindent` should be called. This is for instance the case if `\NeedIndentfalse`, and that parameter is not overruled by `\needindenttrue`.

```
\ifNeedIndent
    \ifneedindent
    \else \removeindentation
\fi \fi
```

Indentation should also be removed in case `\NeedIndenttrue`, but when that parameter is overruled by `\needindentfalse`.

```
\ifNeedIndent
\else \ifneedindent
      \else \removeindentation
\fi   \fi
```

Next we should make sure that `\ifneedindent` is used only for exceptional cases: if the user or a macro sets this parameter to a different value from `\ifNeedIndent`, then that should be obeyed exactly once.

```
\ifNeedIndent
      \ifneedindent
      \else \needindenttrue \fi
\else \ifneedindent \needindentfalse
\fi   \fi
```

This is then the full definition of `\everypar`:

```
\everypar={\controlledindentation}
\def\controlledindentation
   {\ifNeedIndent
          \ifneedindent
          \else \removeindentation
                \needindenttrue
          \fi
    \else \ifneedindent
                \needindentfalse
          \else \removeindentation
    \fi   \fi}
```

Another implementation would be possible:

```
\def\controlledindentation
   {\ifneedindent
    \else \removeindentation \fi
    \let\ifneedindent=\ifNeedIndent}
```

This saves one conditional, but for most paragraphs it involves an unnecessary `\let` command.

2.2 Usage

My aim in developing this indentation scheme was to hide all commands pertaining to indentation in macros. The user should have to specify only once whether paragraphs should indent as a rule:

```
NeedIndenttrue
```

and then macros should declare the exceptions:

```
\def\section#1{...
    \needindentfalse
    ...}
```

2.3 But couldn't you simply ... ?

Maybe people who read this have written macros themselves that end like

```
\def\section#1{...
    ...
    \noindent}
```

or

```
\def\section#1\par{...
    ...
    \noindent}
```

This works reasonably well, but it is not completely safe. In the first case there shouldn't be an empty line after a

```
\section{...}
```

call, and in the second case there can only be one empty line after

```
\section ...
```

The reason for this is that *every* empty line generates a `\par` command, which annuls the effect of the `\noindent`. Hence the more drastic approach.

An argument the other way around can also be found, by the way. As Ron Whitney pointed out to me, the following piece of code causes trouble:

```
\section{Title}%
  {\smallcaps The first} words are ...
```

Any changes made by `\everypar` are now effected *inside a group*. In this case one remedy is to insert a `\leavevmode` command, or to define

```
\def\smallcapswords#1{\leavevmode
   {\smallcaps #1}}
```

which can be used at any place.

Another remedy would be to let all assignments controlling indentation be global. However, there are some subtle objections to this.

3 About macro packages and users

Above I remarked that plain TEX does not use `\everypar`, and that LATEX redefines it a lot. This means that in plain TEX the user is free to take every value of `\everypar` that he or she likes; in LATEX every attempt of the user to use `\everypar` is immediately thwarted.

One might ask how the use of `\everypar` that I have sketched compares to this. Can the user be allowed to access `\everypar`, even if the macro package needs it all the time?

In my own 'Lollipop' format I have taken the following way out. The user or the style designer is allowed to fill in `\everypar`, as long the statement

```
\the\everyeverypar
```

is included. Here `\everyeverypar` is the token list with the constant actions such as indentation control that should be performed always.

A format designer who wishes to hide even this from the user or the style designer, could use the following piece of code

```
\newtoks\temppar
\def\everyparagraph
   {\afterassignment\xevpar
    \temppar}
```

```
\def\xevpar
   {\edef\act{\everypar=
         {\the\temppar
          \the\everyeverypar
         }}%
    \act}
```

so that it becomes possible to write

```
\everyparagraph={\DoSomething
    \everyparagraph={}}
```

while the `\everypar` will still contain all of the constant actions.

Short explanation: `\everyparagraph` is a macro that is made to look like a token parameter by the use of `\afterassignment`. This latter command sets aside the token `\xevpar` for insertion after the first assignment occurring; then whatever follows is assigned to `\temppar`. After this assignment the macro `\xevpar` unwraps the `\temppar` token list and the constant actions into `\everypar`.

4 Conclusion

In a systematic layout indentation commands need never be typed by the user; they can all be hidden in macros.

Using `\everypar` it is possible to prevent indentation both in single instances, and throughout the document. This has the advantage that is is not necessary to zero the `\parindent` parameter or use `\indent` and `\noindent` instructions.

The approach of employing `\everypar` as sketched above can also be used for a paragraph skip scheme, as I will show in a subsequent article.

References

[1] Donald Knuth, The TEX book, Addison-Wesley Publishing Company, 1984.

[2] Leslie Lamport, LATEX, a document preparation system, Addison-Wesley Publishing Company, 1986.

[3] Victor Eijkhout, Unusual paragraph shapes, Tugboat vol. 11 (1990) #1, pp. 51–53.

[4] Stanley Morison, First principles of typography, Cambridge University Press, 1936.

[5] J. Braams, V. Eijkhout, N.A.F.M. Poppelier, The development of national LATEX styles, TUGboat vol. 10 (1989) #3, pp. 401–406.

BIJLAGE DD

A parskip scheme

Victor Eijkhout
University of Illinois at Urbana-Champaign
305 Talbot Lab
104 S. Wright Street
Urbana, Illinois 61801, USA
u641001@HNYKUN11

While I was working on the LaTeX styles described in [1], it became apparent to me that lots of people are rather fond of the sort of layout that can be described as

```
\parindent=0cm
\parskip=6pt % or other positive size
```

Unfortunately, most of them realize this layout by no more sophisticated means than simply inserting these two lines at the beginning of the input. The drawback of such a simple action is that all sorts of vertical spaces are augmented by the `\parskip` when there is absolutely no need to, or where it is positively unwanted. Examples of this are the white space below section headings, and the white space above and below list environments in LaTeX.

In this article I will present an approach that unifies the paragraph skip and the white spaces surrounding various environments. Since the macros given below make use of the `\everypar` token list, this article may be seen as a sequel to an earlier paper on an indentation scheme [2], which is based on a similar principle. The `\everypar` parameter was explained there.

\parskip

TeX starts a paragraph when it switches from vertical to horizontal mode. The vertical mode may have been initiated by a `\par` (for instance because of an empty line after a preceding paragraph) or by a vertical skip command; the switch to horizontal mode can be effected by, for example, a character or a horizontal skip command (see the list in [3, p. 283]). Immediately above the first line of the paragraph TeX will then add glue of size `\parskip` to the vertical list[1].

Apparently, then, the `\parskip` parameter is very simple to use. That this is only an apparent simplicity becomes clear in a number of instances.

For instance, unless precautions are taken, the white space below headings is augmented by the paragraph skip. Precautions against this are not particularly elegant; the easiest solution is to include a

```
\vskip-\parskip
```

statement, to backspace the paragraph skip in advance. Such an approach, however, is somewhat error-prone. Vertical spacing will be messed up if what follows is not a paragraph, but a display formula or a box.

Similar considerations apply to the amounts of white space that surround, for example, list environments, as in LaTeX.

Paragraph skip: to be or not to be

(This section is something of a footnote to the rest of the article. Readers who are not interested in layout consideration may skip the rest of it.)

Ordinarily in plain TeX and in LaTeX the paragraph skip is set to `0pt plus 1pt`, which gives pages some 'last resort' stretchability. However, even an amount of vertical space as small as one point may become very visible, and often without need (see for instance the first page of the preface of [3]).

Furthermore, Stanley Morison states that not indenting paragraphs is 'decidedly an abject method' [4]. However, reading his intention instead of his words, he is only concerned with the recognizability of the individual paragraphs. The positive value of the paragraph skip is sufficient to ensure this. If a layout is based on zero values for both `\parindent` and `\parskip`, one may for instance give the `\parfillskip` a positive natural width to prevent last lines of a paragraph from almost, or completely, lining up with the right margin.

Neither Donald Knuth nor Leslie Lamport seem to have given much thought to the case where the paragraph skip has a positive natural width. Leslie Lamport dismisses all potential difficulties with the remark that 'it is customary not to leave any extra space between paragraphs' [5, p. 94].

[1] Unless this paragraph is at the start of a vertical list, for instance at the start of a vertical box or insertion item.

Environments and white lines

Given that the paragraph skip appears to interact with explicit vertical spacing in user macros, it may seem like a good idea to find a unified approach to both. In the rest of this article I will describe the implementation of the following basic idea: *give the paragraph skip the value zero whenever you do an explicit vertical skip*.

For the presentation I assume a context with some form of environments. These are the assumptions that I make about such environments:

- An environment is a portion of material that is vertically separated from whatever is before and after it. Thus, according to this definition, a portion of a paragraph cannot be an environment, nor can an environment start or end in the middle of a paragraph.
- An environment has associated with it three glue parameters: to an environment `foo` correspond `\fooStartskip` (glue above the environment), `\fooParskip` (the paragraph skip inside the environment), and the `\fooEndskip` (glue below the environment).
- At the outset of the environment a `\StartEnvironment{foo}` statement is executed; at the end of the environment a macro `\EndEnvironment{foo}` is executed. These statements are assumed to contain a `\begingroup` and `\endgroup` respectively.

Such assumptions are sufficiently general for the macros below to be adaptable to existing macro packages. At first sight it would appear as if section headings are not covered by the above points. However, there is no argument against the start and end of an environment occurring in the same macro.

Tools

First I will present two auxiliary macros: `\csarg` and `\vspace`.

The command `\csarg` is only needed inside other macros; it is meant to enable constructs such as

```
\csarg\vskip{#1Parskip}
```

Its definition is

```
\def\csarg#1#2{\expandafter
    #1\csname#2\endcsname}
```

By way of explanation of this macro, consider a simple example. Let us assume that there exists a macro

```
\def\startlist#1{ ...
    \csarg\vskip{#1Startskip}
    ...}
```

The call

```
\startlist{enumerate}
```

will then lead to the following call to `\csarg`:

```
\csarg\vskip{enumerateStartskip}
```

This expands to

```
\expandafter\vskip
 \csname enumerateStartskip\endcsname
```

Now the `\expandafter` forces `\csname` to be executed before the `\vskip`, so the next step of the expansion looks like

`\vskip\enumerateStartskip` and this statement can simply be executed.

Next I need a generalization of `\vskip`, which I will call `\vspace`: a number of calls to `\vspace` should have the effect that only the maximum argument is placed.

```
\newskip\tempskipa
\def\vspace
 #1{\tempskipa=#1\relax
    \ifvmode \ifdim\tempskipa<\lastskip
        \else \vskip-\lastskip
              \vskip\tempskipa
        \fi
    \else \vskip\tempskipa \fi}
```

This may need some explanation too. First, by the assignment

```
\tempskipa=#1
```

I allow the argument of `\vspace` to be both a control sequence, for instance `\parskip`, and a denotation, for instance `5pt plus 3pt`. If one omits the assignment, the latter option would cause trouble in the `\ifdim` test.

The decision to keep the maximum value of the skip, instead of always replacing the last skip, was motivated by phenomena such as a display formula at the end of a list. If the skip below the display is larger than the vertical glue below the list (which may for instance be zero), the former should be retained entirely.

Note that this macro can insert vertical space of size zero. This will for instance happen if it follows a `\par` command at the end of a paragraph. It is then called in vertical mode, but the last item on the vertical list is a box (the last line of the paragraph) instead of a glue item. The parameter `\lastskip` will then be zero.

The environment macros

In this section, I will give the implementation of the macros `\StartEnvironment` and `\EndEnvironment`.

There is a remarkable similarity between these two macros. As I explained above, the basic idea is to have only explicit spacing above and below the environment; thus, the value of `\parskip` should then be zero both for the first paragraph in the environment, and for the first paragraph that follows it. Both macros should then

- save the current value of the paragraph skip;

- set the paragraph skip to zero;
- give TEX a signal that, somewhere in the near future, the old value of `\parskip` is to be restored.

For this I allocate a skip register and a conditional:

```
\newskip\TempParskip
\newif\ifParskipNeedsRestoring
```

The basic sequence for both macros is then

```
\TempParskip=\parskip
\parskip=0cm\relax
\ParskipNeedsRestoringtrue
```

For both macros, however, this sequence needs to be refined slightly.

The paragraph skip to be 'restored' at the start of the environment is the specific value associated with that environment. This gives us:

```
\def\StartEnvironment
 #1{\csarg\vspace{#1Startskip}
    \begingroup %% make changes local
    \csarg\TempParskip{#1Parskip}
    \parskip=0cm\relax
    \ParskipNeedsRestoringtrue}
```

Note that the statement

```
\csarg\TempParskip{#1parskip}
```

denotes an assignment (without an optional equals sign) here.

At the end of the environment we have to be more careful. It may be the case that the environment being ended was inside another environment, and occurred before the first paragraph inside that environment. In that case the value of `\parskip` is zero, and the proper value must still be restored. Therefore, no further actions are required. We arrive at the following implementation:

```
\def\EndEnvironment
 #1{\csarg\vspace{#1Endskip}
    \endgroup %% restore global values
    \ifParskipNeedsRestoring
    \else \TempParskip=\parskip
          \parskip=0cm\relax
          \ParskipNeedsRestoringtrue
    \fi}
```

Note that both macros start with a vertical skip. This prevents the `\begingroup` and `\endgroup` statements from occurring in a paragraph. On a side note: since these macros are executed in vertical mode, I have not bothered to terminate any lines with comment signs. Any spaces generated by these macros are ignored in vertical mode.

Paragraph skip restoring

So far, I have ignored one important question: how exactly is restoring the paragraph skip implemented. For this I use the `\everypar` token list. Basically then, the idea is the same as in [2]: the occurrence of a paragraph will automatically have TEX perform, through the insertion of the `\everypar` tokens, the actions necessary for subsequent paragraphs.

```
\everypar=
   {\ControlledIndentation
      %see Tugboat 11, #3
    \ControlledParskip}
\def\ControlledParskip
   {\ifParskipNeedsRestoring
       \parskip=\TempParskip
       \ParskipNeedsRestoringfalse
    \fi}
```

The cost of a controlled paragraph skip is then one conditional per paragraph. Conceivably, this cost could even be reduced further (to almost zero) by defining

```
\def\CPS % Controlled Parskip
   {\ifParskipNeedsRestoring
       \parskip=\TempParskip
       \ParskipNeedsRestoringfalse
       \let\ControlledParskip=\relax
    \fi}
```

and including a statement

```
\let\ControlledParskip=\CPS
```

in both the `\StartEnvironment` and `\EndEnvironment` macros, and at the start of the job (for instance by including it in the macro package). This approach, however, does not particularly appeal to me. Too much 'pushing the bit around'.

Conclusion

The `\parskip` parameter is arguably the most tricky parameter of TEX. Its workings are very easy to describe, but in actual practice difficulties arise. In this article I have described how treatment of the paragraph skip can be integrated with the glue above and below environments. As in an earlier article on indentation, I use for this the `\everypar` parameter as an essential tool.

References

[1] J. Braams, V. Eijkhout, N.A.F.M. Poppelier, The development of national LATEX styles, TUGboat vol. 10 (1989) #3, pp. 401–406.

[2] Victor Eijkhout, An indentation scheme, TUGboat vol. 11 (1990) #4.

[3] Donald Knuth, The TEX book, Addison-Wesley Publishing Company, 1984.

[4] Stanley Morison, First principles of typography, Cambridge University Press, 1936.

[5] Leslie Lamport, LATEX, a document preparation system, Addison-Wesley Publishing Company, 1986.