An update on the babel system*

Johannes L. Braams

PTT Research Neher laboratories
P.O. Box 421
2260 AK Leidschendam
j.l.braams@research.ptt.nl

Abstract

This article describes the changes that have been made to the babel system, since the article describing the system appeared in *TUGboat* Volume 12, number 2. This article announces the release of a new version of the babel system.

1 Introduction

Since the publication of the babel system in *TUGboat* [1] several changes have occurred. With the new release of LaTeX – which appeared at the end of 1991 – the internationalised version ILaTeX, prepared by Joachim Schrod [2] was withdrawn. But some of its functionality was still needed, so a modification of the babel system was necessary.

Besides this a couple of bugs were reported and had to be fixed. The major problem was that the language changing commands were not 'local', they contained global definitions. In the current version these commands obey grouping correctly.

Some macros that formerly were in language-specific files have been moved to the core of the system, because they are being used in several language-specific files.

2 Changes to the core of babel

The changes to the core of the babel system are the most extensive.

\selectlanguage

The babel user-command `\selectlanguage` now also accepts a control sequence as its argument. This was included to provide compatibility for users who were used to the syntax of the original `german.tex`, but wanted to switch to babel. The escape character is 'peeled off' and the name of the control sequence is then used as the name of the language to select.

Another change to the `\selectlanguage` macro is that it now stores the name of the current language in the control sequence `\languagename`. The contents of this control sequence could be tested in the following way:

```
\edef\tmp{\string english}
\ifx\languagename\tmp
  ...
\else
  ...
\fi
```

The construction with `\string` is necessary because `\languagename` returns the name with characters of category code `12` (other).

Saving macro definitions

A new way of handling macros that are temporarily *re*defined was developed by Bernd Raichle and included in the core of the babel system. Two new macros, for use in the language-specific files have been introduced.
These macros, `\babel@savevariable`⟨*register*⟩ and `\babel@save`⟨*macro*⟩, append code to `\originalTeX`. This code restores the value (or meaning) of what was saved when `\originalTeX` was executed.

Special characters

Some of the language-specific files introduce one or more characters that are special in some way. Such characters have to be added to `\dospecials` (and `\@sanitize` too for LaTeX) whenever their special meaning is activated. But, they may have to be removed again when another language, which doesn't use them, is in effect.

To this end two new control sequences, that are meant to be used in the language-specific files are introduced. They are `\babel@add@special` and `\babel@remove@special` and perform the necessary tasks.

*Published in TUGboat **14.1**, 60–62 (1993).

Additional facilities

A specific request from Joachim Schrod for babel, was the possibility to extend the definition of a control sequence on the fly. It should, for instance, be possible that the user adds a macro of his own to the definition of `\extrasenglish`.

This feature is now provided by the macro `\addto{`⟨*control sequence*⟩`}{`⟨*TEX code*⟩`}`. It is now used throughout the language-specific files to build the macros `\extras`⟨*lang*⟩ and `\noextras`⟨*lang*⟩.

The support macros `\allowhyphens`, `\set@low@box` and `\save@sf@q` have been moved from the language-specific files to the core of the babel system.

The files

In the previous release a file called `latexhax.com` was provided. This was needed to provide some macros normally defined by LaTeX, to `plain` TeX users. The need for this file has been removed in the current release of the babel system.

In the previous release of the system, four different files were provided (all derived from `hyphen.doc` that were needed for different combinations of versions of TeX and `plain.tex` or `lplain.tex`. This has been changed. In the current version only two different files are derived from `hyphen.doc`. They are `babel.switch` and `babel.hyphen`. The file `babel.switch` is needed for people who can't build a new format or don't have TeX version 3. The file `babel.hyphen` should be loaded into the format by iniTeX. It provides the macros from `babel.switch`, but additionally it reads the file `language.dat`, which specifies the languages for which hyphenation patterns should be loaded.

In the previous release the file `babel.com` contained redefinitions for a lot of LaTeX macros to replace texts with control sequences. This has been removed, because it is no longer necessary for releases of LaTeX dated december 1991 or later. Those who still have an older release of LaTeX can produce a special version of `babel.com` by including the docstrip option ⟨*names*⟩ when stripping the file `babel.doc`.

With the release of the new version of Frank Mittelbach's doc package the stripped files are no longer distributed. The distribution of babel now includes a file `install.babel` with which you can produce them (give the command `tex install.babel`).

3 Changes to the language specific files

Bernd Raichle has invented a solution for things like `\char"45` when the `"` is active. His solution (from german 2.3e) has been included in germanb and is copied for other language specific files that have an active `"`.

A few terms have been added to the `\captions`⟨...⟩ macros, again following `german.tex`. These terms are `\prefacename`, `\seename` and `\seealsoname`. I don't have the correct translations for all languages yet, but that will be repaired as soon as someone provides them to me.

For the dutch language the behaviour of the active double quote has been slightly modified. It has been noted that there is a difference between `"e`, where a 'trema' should be produced and `\"u`, where we should get an 'umlaut'. The difference between the two is that the 'trema' should disappear at a hyphenation point, whereas the 'umlaut' should not.

References

[1] Johannes Braams, *Babel, a multilingual style-option system for use with LaTeX's standard document styles*, *TUGboat* 12 (1991) #2, p. 291–301.

[2] Joachim Schrod, *International LaTeX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.

Herziene afbreekpatronen voor het Nederlands

Piet Tutelaers

Technische Universiteit Eindhoven
Rekencentrum
`rcpt@urc.tue.nl`

1 Waarom nieuwe afbreekpatronen?

De CELEX patronen van Henk Penning (Rijksuniversiteit Utrecht, `henkp@cs.ruu.nl`) zijn de meest gebruikte voor het Nederlands. Ze zijn beschikbaar op alle goede TEX-fileservers zoals op `ftp.cs.ruu.nl`. Deze afbreekpatronen hebben echter de volgende bezwaren:

1. werken niet goed voor woorden met diacritische tekens (8-bits karakters)
2. breken niet af in de eerste en laatste twee letters van een woord
3. voldoen niet aan de nieuwere afbreekregels uit de 'Herziene Woordenlijst Nederlandse taal' (Groeneboekje boekje 1990, in het vervolg afgekort als GB90).

Feitelijk was punt 1 voor mij de aanleiding om de bestaande patronen te herzien. TEX3.0 biedt namenlijk de mogelijkheid om te werken met 8-bits karakters en 8-bits fonts (Extended Computer Modern bijvoorbeeld). Wil je daar optimaal profijt van hebben dan heb je uiteraard ook afbreekpatronen nodig waarin deze 8-bits karakters voorkomen. Met de huidige CELEX patronen wordt `Curaçaoënaar` afgebroken als `Cu-raça-oë-naar`. Volgens het GB90 is `Cu-ra-çao-e-naar` de correcte spelling. We zullen later zien waarom `Cu-ra-çaoë-naar` het beste is wat we met TEX3.0 kunnen bereiken.

Bij het maken van de CELEX patronen zijn woorden waarin diacritische tekens voorkomen weggelaten omdat TEX toendertijd alleen ASCII codes kon verwerken. Je kunt je afvragen hoe slecht deze 7-bits patronen voor woorden met een diacritisch teken zijn. In het door mij gebruikte woordenboek komen 2090 woorden voor waarin een trema of accent aanwezig is. Deze woorden bevatten in totaal 4774 afbreekplaatsen. Hiervan worden er 54 in de verkeerde lettergreep geplaatst (1.2%) en 497 (10.4%) gemist. In het algemeen breken de CELEX patronen niet af voor een trema op 30 gevallen na. In de volgende voorbeelden zijn deze plaatsen aangegeven met een '.':

```
afge.ëist
associ.ëer
mede.ëter
mede.ïngezetene
shampoo.ën
```

De overige 24 fouten treden op in verschillende woordtypen zoals:

```
associ.és
Curaça.oënaar
ing.rediënt
procédee.tje
vol.tampère
```

Meestal wordt ontraden om woorden af te breken in de eerste twee en laatste drie letters van een woord. Er kunnen echter bijzondere omstandigheden zijn, bijvoorbeeld als je met erg smalle tekstkolommen werkt, waardoor je van deze hoofdregel wilt kunnen afwijken. Je kunt dan `\lefthyphenmin` en `\righthyphenmin` op de gewenste waarden zetten.

De afbreekregels in het GB90 zijn ten opzichte van de regels uit het Groene boekje 1954 op een aantal punten vereenvoudigd. Bastaardwoorden worden zoveel mogelijk via de Nederlandse regels afgebroken (hockey-en in plaats van hock-ey-en, cros-sen in plaats van cross-en). Er zijn echter ook een aantal veranderingen ingevoerd met het doel om de afbreekregels te vereenvoudigen:

GB54	GB90
ab-er-ra-tie	a-ber-ra-tie
ad-o-ra-tie	a-do-ra-tie
ad-e-quaat	a-de-quaat
il-lus-tra-tie	il-lu-stra-tie
leeuw-e-rik	leeu-we-rik
pres-crip-tief	pre-scrip-tief
pres-crip-tie-ve	pre-scrip-tie-ve
pros-pec-tief	pro-spec-tief
pros-pec-tie-ve	pros-pec-tie-ve
re-gis-ter	re-gis-ter
re-gis-tra-tie	re-gi-stra-tie
ver-nieuw-end	ver-nieu-wend

2 Afbreekregels in TEX

Wanneer TEX, in een poging om een mooie rechterkantlijn te maken, genoodzaakt is om een woord af te breken dan gaat dit met de volgende prioriteiten:

1. op de plek van een expliciete afbreekplaats (`\discretionary`)
2. volgens een uitzonderingsgeval (`\hyphenation`)
3. volgens de algemene afbreekregels (`\patterns`).

2.1 Expliciete afbreekplaats

Het Nederlands heeft een aantal situaties die netjes op te lossen zouden zijn met het `\discretionary` commando. Denk maar aan woorden als `menuutje` (`me-nu-tje`), `procédeetje` (`pro-cé-dé-tje`) en `Curaçaoënaar` (`Cu-ra-çao-e-naar`). De praktische bruikbaarheid van het discretionary commando in TEX is echter zeer beperkt omdat je het niet kunt gebruiken in uitzonderingsgevallen en patronen. Wil je de genoemde woorden volgens de Nederlandse taalregels afbreken dan rest er niets anders dan de woorden kompleet in te voeren:

```
menuutje
  me\-nu\discretionary{-}{}{u}tje
procédeetje
  pro\-cé\-d\discretionary{é-}{}{ee}tje
Curaçaoënaar
  Cu\-ra\-çao\discretionary{-}{e}{ë}naar
```

Nu kun je de pijn wel proberen te verzachten met macros als:

```
\def\uu{u\discretionary{-}{u}{u}}
\def\ee{\discretionary{é-}{}{ee}}
\catcode`\"==\active % Babel30d methode!
\def"#1{\allowhyphens\discretionary{-}
   {#1}{\"{#1}}\allowhyphens}
```

Als je nu de woorden invoert als:

```
men\uu tje
procéd\ee tje
Curaçaoënaar
```

dan worden deze woorden alleen afbroken op de plaatsen waar de discretionaries voorkomen (aangegeven met '`*`'). En dus niet op alle andere plaatsen (aangegeven met '`-`'):

```
me-nu*tje
pro-cé-dé*tje
Cu*ra-çao*e-naar
```

De `\allowhyphens` in de `"`-macro zorgt ervoor dat een tremawoord wordt opgedeeld in drie onafhankelijke delen. Het deel vóór het trema, het trema zelf en het deel ná het trema. Hierdoor worden in het algemeen meerdere afbreekplaatsen gevonden. Er zijn echter ook gevallen waarin deze benadering tot fouten leidt, zoals in:

```
ar*chaï-see.rde
ar*chaï-see.r*den
be*doeïe-n.en*tent
be*doeïe-n.en*ten*ten
geë-ve-n.aard
on-geë-ve-n.aard
on-geë-ve-n.aarde
```

Normaal wordt niet afgebroken binnen de eerste `\lefthyphenmin` en de laatste `\righthyphenmin` karakters van een woord. Wanneer zich echter in een woord een `\discretionary`-commando bevindt dan wordt van deze regel afgeweken. Woorden als `België` en `aëroob` worden ten onrechte afgebroken voor het e-trema. In de `"`-macro wordt bovendien geen onderscheid gemaakt tussen een trema en een umlaut. Woorden als `fröbelen` en `maïs` gaan dan ook fatikaal fout.

Het Nederlands kent ook woorden waarin een apostroph wordt gebruikt (`baby'tje` en `juffertje-in-'t-groen`). Deze woorden worden niet altijd goed afgebroken (`baby.'.tje`).

2.2 Uitzonderingen

Woorden die met de afbreekpatronen foutief worden afgebroken kunnen afzonderlijk worden behandeld. Dit kan voor de duur van één document (preamble van LATEX) of voor alle LATEX-runs als je deze uitzonderingen met IniTEX meevertaalt.

De file `GB90.b+m` bevat de woorden uit de gebruikte woordenlijst die fout gaan met `GB90.8pat`. Plaatsen aangegeven met '`.`' gaan fout (er komt ten onrechte een afbreekstreepje), plaatsen met '`*`' aangegeven gaan goed en plaatsen gemarkeerd met '`-`' worden gemist.

Uit `GB90.b+m` zijn de volgende 'exceptions' gedestilleerd:

```
% Exception list for GB90.8pat (July '93)
\hyphenation{
aan-pers-te
acht-en-der
acht-en-ders
acht-en-der-tig
acht-en-der-tig-ste
ant-arc-tis
be-scherm-en-gel
be-scherm-en-ge-len
don-der-aal
drie-ster
gast-rol-len
ge-laats-trek-ken
han-dels-taal
ket-ting-ste-ken
lands-taal
lui-ste
mi-nis-ters-por-te-feuil-le
mi-nis-ters-por-te-feuil-les
moet-je
pa-ling-ste-ken
schel-linkje
spie-gel-ei
ti-chel-aar-de
vier-en-der-tig
vier-en-der-tig-ste
}
```

2.3 Afbreekpatronen

De patronen gebaseerd op het GB90 zijn gemaakt inclusief de diacritische tekens. Vanwege de problemen die optreden met het `\discretionary` commando worden woorden niet afgebroken voor tremas. Onregelmatige woorden (`menuutje`) en woorden met verschillende betekenissen (`buur-tje` en `buurt-je`, `wets-taal` en `wet-staal`) worden ook niet afbroken op plaatsen waar deze afbreking niet eenduidig is.

De patronen zijn gemaakt met `patgen2` (een 8-bits versie van `patgen`) uitgaande van een woordenlijst die is afgeleid van het Groeneboekje '54. Deze woorden zijn afgebroken met de CELEX patronen en vervolgens in een aantal slagen met de hand aangepast aan de spelling van GB90 (zeer tijdrovend!). Wanneer het mij lukt om alsnog de officiële woorden van het GB90 te lenen van het Instituut voor Lexicologie dan zal ik de patronen opnieuw berekenen. Tot dan blijven deze patronen van kracht.

De gebruikte woordenlijst bevat 163861 woorden met in totaal 361780 afbreekplaatsen. De meeste woorden stammen uit het GB54. Ook de woorden uit het CELEX bestand die fout werden afgebroken met de CELEX patronen of waarin bepaalde afbreekplaatsen gemist werden, zijn aan deze lijst toegevoegd (plus minus 2000). In tabel 1 zijn de patgen[1] resultaten bij elkaar geplaatst. Zie [7] voor de betekenis van de waarden uit de eerste drie kolommen.

level	lengte	parameters	patronen	goed	fout	mis
1	1(1)4	1 2 20	832	97.48	11.13	2.52
2	1(1)4	2 1 4	1196	92.29	0.55	7.71
3	1(1)5	1 1 1	3672	99.86	3.87	0.14
4	1(1)6	3 2 1	2396	87.87	0.01	2.13
5	1(1)8	1 1000 1	2167	100.00	0.01	0.00
					(24)	(7)

Tabel 1: *Patgen resultaten van GB90.8pat*

	GB90	CELEX (GB54)
patterns	11803	12698
trie ops	185	188
trie words	9985	10335
bytes	52113	51748

Tabel 2: *Geheugen beslag van CELEX en GB90 patronen*

De resultaten zijn zeer goed. Het aantal foutieve afbreekplaatsen is maar 24 terwijl slechts 7 plaatsen gemist worden. De CELEX patronen leverden 210 (0.02%) foutieve en 5877 (0.49%) gemiste afbreekplaatsen op. Daarbij moet worden opgemerkt dat het CELEX bestand fouten bevatte die niet zijn gekorrigeerd.

De afbreekpatronen zijn zo ontworpen dat ze een minimaal geheugenbeslag leggen op TeX. Als we de CELEX patronen en de nieuwe GB90 patronen vergelijken dan zien we uit tabel 2 dat dat onderling weinig verschilt.

3 Conclusie

Het model dat door TeX wordt gebruikt voor het afbreken van woorden levert voor het Nederlands zeer acceptabele resultaten op. Echter woorden waarin een trema voorkomt of woorden met een veranderende lettergreep kunnen niet goed worden behandeld vanwege de beperkingen die TeX oplegt aan het discretionary commando. Omdat de methode die nu gebruikt wordt in `dutch.sty` allerlei foute resultaten geeft, zijn bij het genereren van de herziene patronen afbreekplaatsen voor tremas, in veranderende lettergrepen en in woorden waarin de afbreking niet eenduidig is, weggelaten. Woorden met een koppelteken erin worden alleen op die plaats afgebroken omdat dit impliciet wordt afgebeeld op een discretionary. Wanneer zo'n koppelteken lange woorden verbindt dan worden die niet afgebroken zoals 'juffertje' in 'juffertje-in-'t-groen' en 'tolletje' in 'a-al-tolletje' En er nog geen goede oplossing voor woorden met een apostroph.

Samengevat:

- vóór een trema kan beter niet worden afgebroken
- woorden waarvan de afbreking onregelmatig is (veranderende lettergreep) kunnen beter niet worden af-

[1] Een run op een DEC/OSF1 alpha kost ongeveer 45 minuten rekentijd. Op mijn 486DX/33Mhz systeem thuis met 386BSD UNIX 1 uur en 17 minuten.

gebroken in die lettergreep
- woorden waarin koppeltekens '-' voorkomen worden alleen op die plaatsen afgebroken
- woorden met een apostroph worden niet altijd goed afgebroken.

4 De beschikbaarheid

Realiseer je dat deze patronen alleen zin hebben als je overstapt naar 8-bits fonts. Je kunt dan in je LaTeX invoer rechtstreeks een i-trema plaatsen zonder dat je daarvoor het 7-bits `\"{\i}` commando gebruikt. Diverse systemen ondersteunen ISOlatin1 zodat je deze tekens met je editor rechtstreeks kunt invoeren.

Wil je de patronen eens uit proberen? Ze zijn beschikbaar via ftp op `ftp.urc.tue.nl` in `tex/8bit/newhyph`. Ook als je ze niet onmiddellijk in LaTeX wilt gaan gebruiken, kun je ze uitproberen. Er is namelijk een programma `hyphenate.c` bij aanwezig waarmee je interaktief woorden kunt afbreken. Dit programma accepteert als argument de naam van een file waarin zich afbreekpatronen en uitzonderingen bevinden en bepaalt van de ingevoerde woorden hun afbreekplaatsen. Met de optie `-L`*left* en `-R`*right* kun je opgeven hoeveel letters aan het begin en eind van een woord niet mogen worden afgebroken. Deze waarden zijn standaard ingesteld op *left*=`2` en *right*=`3`. Met de `-d` optie krijg je extra tussenresultaten zoals de patronen die zijn gebruikt voor het berekenen van de afbreekplaatsen (`-d1`). Hier is een kleine demonstratie van de mogelijkheden van `hyphenate` [2]:

```
hyphenate -d1 -L1 -R1 GB90.8pat
Curaçaoënaar
3cu
2ur
u1ra
1ç
a3o
4oë
4ë
ë3na
a4a4
4ar.
4r.
.3C2u1ra1ça4o4ë3na4a4r.
Cu-ra-çaoë-naar
```

In `tex/8bit/newhyph` tref je de volgende files aan:

```
(8) patronen.tex
      (deze file, voor de verwerking heb
       je NFSS en de DC-fonts nodig)
    patronen.ps.z
      (PostScript uitvoer van
       patronen.tex)
(8) GB90.8pat
      (de 8-bits patronen)
    GB90.7pat
      (de 7-bits gedaante van GB90.8pat)
(8) GB90.b+m
      (bad+missed woorden)
(8) WdNT.G-B.z
      (de gebruikte woor-den-lijst)
    patgen.out
      (de log van patgen)
    patgen.in
      (de invoer waarden van patgen)
(8) dutch.tra
      (de transition file voor patgen)
    dcdutch.sty
      (een 8-bits versie van dutch.sty;
       zorgt ervoor dat voor tremas niet
       wordt afgebroken)
(8) test.tex
      (een plain TeX test file; legt
       diverse problemen van het
       afbreken in het Nederlands bloot!
       Voor de verwerking heb je NFSS en
       de DC-fonts nodig)
    src
      (deze directory bevat de sources
       van een aantal hyphenation
       tools zoals hyphenate.c)
    src/README
      (bevat nadere gegevens hierover)
```

De files voorafgegaan door een (8) bevatten 8-bits karakters. En de files met achtervoegsel `.z` zijn gecomprimeerd met `gzip`.

References

[1] Peter Breitenlohner. *The eight bit patgen extensions.* Unpublished, patgen2.web sources

[2] *CELEX woordenbestand.* Centre for Lexical Information. Universiteit van Nijmegen

[3] Donald E. Knuth. *Computers and Typesetting Vol. A–E.* Addison–Wesley, Reading, MA, 1987–1991

[3a] Vol A: *The TeXbook*, 11. ed. 1991

[3b] Vol B: *TeX: The Program*, 4. ed. 1991

[4] Yannis Haralambous. *Syntax of translation file.* Unpublished, distributed via BITNET

[5] *Herziene Woordenlijst Nederlandse taal*, SDU, Den Haag, 1990, vijfde oplage

[6] Leslie Lamport. *LaTeX – A Document Preparation System* Addison–Wesley, Reading, MA, 1985

[7] Franklin Mark Liang. *Word Hy-phen-a-tion by Com-put-er*, Department of Computer Science, Stanford University, 1983

[8] Norbert Schwarz, *DC-fonts.* Unpublished DC METAFONT sources. Rechenzentrum Ruhr-Universitaet Bochum, Germany

[2] Zie appendix H van het TeXbook voor meer informatie over afbreekpatronen.

Typesetting paragraphs of a specified shape

Donald Arseneau

4694 West 8 Ave,
Vancouver BC,
V6R 2A7 Canada
`asnd@reg.triumf.ca`

(*Editor's note*: This description of an unusual macro file, `shapepar.sty`, is taken from the documentation; the full style file can be found in CTAN archives.)

`\shapepar` is a macro to typeset paragraphs of a specified shape. The total size is adjusted automatically so that the entire shape is filled with text. This is distinct from the normal `\parshape` command which specifies a shape *and* a size, which may be only partially filled, or over-filled, from top to bottom. In a `\shapepar` there can be no displayed math, and no `\vadjust` material, (including `\vspace`). `\Shapepar` (capital S) is just like `\shapepar` except the paragraph is boxed so it cannot be split over two pages. Shaping paragraphs this way is a slow process, so this style is mainly intended for cards, invitations etc., not for whole books! Although short paragraphs process much faster, only long paragraphs accurately fill complex shapes.

These macros work for both LaTeX and plain TeX. For LaTeX, specify `\documentstyle[...shapepar...]`, or for either, `\input shapepar.sty`.

The command `\shapepar` should be used at the beginning of a paragraph, and it applies to the entire paragraph. There is one parameter: a description of the shape, <*shape_spec*>.

```
\shapepar {<shape_spec>} Text of para...
```

The syntax rules for <*shape_spec*> are very specific, and must be followed closely. (In these rules, `{ }` mean explicit braces, `[ ]` denote optional parts, `< >` surround a keyword that is defined (perhaps loosely), and | means 'or'; do not type `[ ]` `< >` or |, only `{ }`.)

```
<shape_spec}> = {<h_center>} <lines>
<lines> = <line_spec> [\\<lines>]
```

That is, the shape is specified as a single number in braces, followed by the specifications for the lines, with the lines separated by `\\`. The final paragraph will have its <*h_center*> position centered on the page. <*h_center*> is a number (like 10.5) of arbitrary units; whatever units are used for lengths and positions in the <*lines*>, they just need to be consistent.

The lines in the spec are not lines of text; nor are they the lines that you would use to draw the shape itself. They are horizontal scans across the shape at irregular intervals. Curved shapes need many scan lines for accurate rendering while simple shapes need few. Draw a shape on paper, then draw a series of horizontal lines across the shape, including lines that just touch the top and the bottom of the figure. Each line crosses over pieces of the figure in some region. These intersections of line and figure define a <*line_spec*>.

```
<line_spec> = {<v_pos>} &lt;segment&gt;
                     [ other &lt;segment&gt;s ]
```

The <*v_pos*> is the vertical position of the line. Each <*line_spec*> must have a position greater than or equal to that of the previous line, and with all <*v_pos*> > -1000. Position is measured from top to bottom, and always moving down. Each <*segment*> represents a region where text will go in the final paragraph; it is the segment of the horizontal scan line that overlaps the body of the figure. There are five types of segment:

```
&lt;segment&gt; = t{pos}{len} | b{pos} |
                          e{pos} | s | j
```

`b{pos}`	begin text at a point at horizontal position pos
`e{pos}`	end text at a point at horizontal position pos
`t{pos}{len}`	make a block of text at position pos with length len
`s`	split text (begin whitespace)
`j`	join two text blocks (end a gap)

The most common type of segment is *t* (text). The other types are degenerate in that they are single points rather than finite segments. Types *s* and *j* have no explicit position, but they must appear between text segments, and those texts should abut; e.g., `t{3}{2}st{5}{4}` (text from 3 to 5 and text from 5 to 9).

Let's jump right into a simple example, and the meanings will be clearer. A 'diamond' shape can have the four vertices:

```
                 (x=1,y=0)
                     .
+---> x
!        (0,1)  .         .  (2,1)
!
V y                  .
                   (1,2}
```

This shape can be exactly specified by just three scan lines passing through the vertices. The specification is:

`{1}%`	*h_center*: $x = 1$
`{0}b{1}\\%`	text block begins at point y=0, x=1
`{1}t{0}{2}\\%`	this scan (at y=1) crosses text (len=2) starting at x=0
`{2}e{1}`	text block ends at point y=2, x=1

Other specification lines, like `{1.5}t{0.5}{1}\\%` could be inserted, but would make no difference–the shape is interpolated linearly between scan lines.

Every block of text must start with a b specifier and end with an e spec. on some line below. Every segment specified by t must have a length greater than zero. If two blocks of text merge to form one (like at the top of a heart shape) there should be a j spec at the point of junction. If one block bifurcates (like at the top of a hole in a doughnut) there should be an s spec.

Thus, the first line for any valid shape description must consist of only b segment descriptors; the last line can only have e type descriptors. Although the definition of the units is arbitrary, the numbers should range in magnitude from $\tilde{}.1 - 100$ to avoid numeric overflows and underflows.

If there are errors in the format of the specification, `\shapepar` might complain with the error message 'Shaped Paragraph Error: Error in specification. Check carefully!' At this point you may as well type `x` or `e`, as there is very little chance that TeX will continue successfully. You might also get one of TeX's regular error messages, like 'Illegal unit of measure (pt inserted).' or 'Missing number, treated as zero.' or you might get no error message at all, just ridiculous formatting. Check shape syntax carefully against the rules and the examples before running them through TeX.

What to do if the figure does not start at a point–if it has a flat top? It can start at a single point, but have the next scan line at the same vertical position! A square paragraph is specified by:

`{1}%`	centerline at x=1
`%`	(x=1 is horizontally centered on page)
`{0}b{0}\\%`	begin at (0,0)
`{0}t{0}{2}\\%`	text at y=0, width=2
`{2}t{0}{2}\\%`	text at y=2, width=2
`{2}e{1}%`	end at (1,2)

Both `\diamondpar` and `\squarepar` are defined as paragraphs with these shapes.

Sit at word processor for two hours composing title for new book. Head for tea room at 11 0'clock where six colleagues are sitting round walls in silence. Reminds me of mental hospital day room but no strait jackets, except perhaps intellectual. Make coffee with back to them in order to surreptitiously use someone else's milk from fridge. Wonder why no one ever asks me what I have been doing in the department for the last three years. Listen to sudden burst of animated conversation about new Mac software, and realise that my social isolation is due to the fact that I have an Amstrad.

Now let's get more ambitious. A heart shape must have two simultaneous beginnings, a short stretch of separate text, ending with a join, thereafter there is just one stretch of text leading to the final bottom point. This shape has many scan lines so that the smooth flowing curves are preserved.

```
\def\heartshape{%
{20}{0}b{13.32}b{26.68}%
\\{.14}t{10.12}{4.42}t{25.46}{4.42}%
\\{.7}t{9.14}{7.16}t{23.7}{7.16}%
\\{1.4}t{8.4}{9.02}t{22.58}{9.02}%
\\{2.1}t{7.82}{10.42}t{21.76}{10.42}%
\\{2.8}t{7.36}{11.58}t{21.06}{11.58}%
\\{3.5}t{6.98}{12.56}t{20.46}{12.56}%
\\{4.2}t{6.68}{13.32}jt{20}{13.32}%
\\{4.9}t{6.48}{27.04}%
\\{5.6}t{6.34}{27.32}%
\\{6.3}t{6.28}{27.44}%
\\{7}t{6.26}{27.48}%
\\{7.7}t{6.27}{27.46}%
\\{8.4}t{6.32}{27.36}%
\\{9.1}t{6.4}{27.2}%
\\{9.8}t{6.52}{26.96}%
\\{10.5}t{6.68}{26.64}%
\\{11.9}t{7.12}{25.76}%
\\{13.3}t{7.72}{24.56}%
\\{14.7}t{8.51}{22.98}%
\\{16.1}t{9.5}{21}%
\\{17.5}t{10.69}{18.62}%
\\{18.9}t{12.08}{15.84}%
\\{20.3}t{13.7}{12.6}%
\\{21.7}t{15.62}{8.76}%
\\{22.4}t{16.7}{6.6}%
\\{23.1}t{17.87}{4.26}%
\\{24.6}e{20}%
}
```

Sit at word processor for two hours composing title for new book. Head for tea room at 11 0'clock where six colleagues are sitting round walls in silence. Reminds me of mental hospital day room but no strait jackets, except perhaps intellectual. Make coffee with back to them in order to surreptitiously use someone else's milk from fridge. Wonder why no one ever asks me what I have been doing in the department for the last three years. Listen to sudden burst of animated conversation about new Mac software, and realise that my social isolation is due to the fact that I have an Amstrad.

Look at `\heartshape` and find the two b specifiers at the beginning; find the j a few lines below. Notice that above the j there are two segments per line, but only one below it; the text to the left and right of the join meet at the join point: 20. I drew this heart freehand, and measured lengths from the sketch, so you should be able to do better!

Text can have holes. For example, a doughnut-shape would have a b on the first line, followed by some lines with a single t, then a line with $t\ s\ t$ at the start of the hole. The hole is represented by lines with two t specs–the gap between them is the hole. A line with $t\ j\ t$ ends the hole. There are more lines with single t, and then an e line to end with. Our final example is a nut. Not a doughnut, but a hex-nut (for a machine screw) — a regular hexagon with a circular hole in the center. The hexagon is flat on top and bottom so the specification begins and ends like the square shape. The circle is rendered as a 24-gon, beginning with a split (s) of the surrounding text and ending with a join (j). If the spacing of the scan lines looks odd, it is because the hexagon alone would need few scans, but the circle needs many; the points on the circle are at 15 degree intervals.

```
\def\nutshape{%
{0}%
{0}b{0}\\%
{0}t{-12.5}{25}\\%
{11.65}t{-19.23}{19.23}st{0}{19.23}\\%
{11.99}t{-19.42}{16.835}t{2.59}{16.835}\\%
{12.99}t{-20}{15}t{5}{15}\\%
{14.58}t{-20.92}{13.85}t{7.07}{13.85}\\%
{16.65}t{-22.11}{13.45}t{8.66}{13.45}\\%
{19.06}t{-23.51}{13.85}t{9.66}{13.85}\\%
{21.65}t{-25}{15}t{10}{15}\\%
{24.24}t{-23.51}{13.85}t{9.66}{13.85}\\%
{26.65}t{-22.11}{13.45}t{8.66}{13.45}\\%
{28.72}t{-20.92}{13.85}t{7.07}{13.85}\\%
{30.31}t{-20}{15}t{5}{15}\\%
{31.31}t{-19.42}{16.835}t{2.59}{16.835}\\%
{31.65}t{-19.23}{19.23}jt{0}{19.23}\\%
{43.3}t{-12.5}{25}\\%
{43.3}e{0}%
}
```

Sit at word processor for two hours composing title for new book. Head for tea room at 11 0'clock where six colleagues are sitting round walls in silence. Reminds me of mental hospital day room but no strait jackets, except perhaps intellectual. Make coffee with back to them in order to surreptitiously use someone else's milk from fridge. Wonder why no one ever asks me what I have been doing in the department for the last three years. Listen to sudden burst of animated conversation about new Mac software, and realise that my social isolation is due to the fact that I have an Amstrad.

`\shapepar` cheats a bit when the horizontal gap between two bits of text is small (like down in the notch of `\heartpar`). When the gap is less than an inter-word space it is eliminated, and the texts are joined; when it is somewhat larger it is expanded to give it more visibility.

Since the processing is slow, there are some messages to say how things are going. These can be eliminated to save space (Put a % at the start of every `\message` line.) Or you can get even more verbose messages by *removing* the % that precedes many other `\message` commands.

There are also a number of parameters which can be changed to affect the size-optimization procedure. Search for the word *optimize* in the source.

TEX from \indent to \par*

Marek Rychko [†] and Bogusław Jackowski [‡]

1 Introduction

There are some seemingly simple questions about TEX which may be difficult to answer without precise knowledge of TEX mechanisms.

In Section 2 we will ask three such questions, encouraging the reader to answer them without reading the explanation.

Actually, the explanation follows immediately from a detailed specification of TEX's action at the beginning and at the end of a paragraph. We believe that if such a specification of all TEX's mechanisms existed, answers to most questions concerning behavior of TEX would be equally simple.

The pivot sections are 5 and 7. Section 3 contains necessary introductory material.

2 Questions

In all questions we assume the normal meaning of tokens of plain TEX.

Q1. What is the difference between:

(*) `\everypar{\def\indent{1}}`
`\indent 3 is a prime number.`
and

(**) `\everypar{\def\vrule{1}}`
`\vrule 3 is a prime number.`
What is typeset in both cases and why?

Q2. Assuming that TEX is in vertical mode, what is the difference between:

(*) `\parindent=0mm \indent\par`
and

(**) `\noindent\par`
What is appended to the main vertical list and why?

Q3. What is the difference between:

(*) `\par`
and

(**) `{\par}`
What is the state of TEX after executing these commands in both cases and why?

3 From input characters to commands

Let us start with a closer look into TEX's way of processing of input data. Three levels of the processing can be distinguished:

L1. Reading characters from the input file and transforming them into tokens (lexical analysis).

L2. Expanding tokens.

L3. Executing commands; at this level TEX creates internal lists (horizontal, vertical and math lists), transforms them into boxes and writes some boxes to the `DVI` file (using the `\shipout` command).

Knuth says about 'eyes,' 'mouth' and 'stomach' of TEX, etc.; we prefer to speak about 'levels.'

3.1 Names and meanings of tokens

In order to understand what happens at the beginning and at the end of a paragraph it is essential to be aware of the difference between names and meanings of tokens.

Following Knuth, we will denote by `*\xyz` the meaning of the command `\xyz` at the beginning of the TEX job. By $\boxed{\texttt{xyz}}$ we will denote a token, the name of which consists of the letters '`xyz`'. Such a token is created by TEX from the sequence of letters '`xyz`' preceded by a current escape character, usually backslash.

For example, the token $\boxed{\texttt{hbox}}$, the *name* of which consists of the letters '`hbox`,' has initially the meaning `*\hbox`. Saying '`\let\hbox=\par`' a user may change the meaning of $\boxed{\texttt{hbox}}$ to the current meaning of $\boxed{\texttt{par}}$, most likely to `*\par`. Incidentally, TEX replaces every empty input line with the token $\boxed{\texttt{par}}$ regardless of the meaning of this token. The meaning of $\boxed{\texttt{par}}$ may be `*\par`, but $\boxed{\texttt{par}}$ may be also, for example, a macro expanding to a sequence of tokens.

3.2 Transforming input characters into tokens

From the point of view of TEX, the input file is a sequence of characters organized into lines. TEX reads such characters one by one and transforms them at the level 1 into so-called tokens, according to definite rules. For example, the following sequence of 15 input characters:

*Published in TUGboat **14.1**, 60–62 (1993).
†Wydawnictwo Do, ul. Filtrowa 1, 00-611 Warszawa, Poland
‡ul. Tatrzańska 6/1, 80-331 Gdańsk, Poland

`\ e n s p a c e ␣ |D o n . . .|`

is transformed into a sequence of 7 tokens:

$\boxed{\texttt{enspace}}$`Don...`

The first one is a control sequence token and the remaining are character tokens stored by TEX along with their category codes.

Each token created at this level is associated with its current meaning which can be either *a primitive meaning* (a meaning that is built in into TEX) or it can be *a macro* (a meaning that can be defined by a user in terms of other meanings). Regarding the meaning we can classify all tokens as follows:

(a) with respect to expandability as *expandable* and *unexpandable*;
(b) with respect to primitivity as *primitive* and *macros*.

The expandable tokens can be primitive, like `\if`, `\the`, `\noexpand`, `\csname`, or they can be macros defined using `\def` or a related assignment (`\edef`, `\gdef`, `\xdef`).

All unexpandable tokens are primitive. This group contains, among others: tokens like `\hskip`, `\hbox`, etc.; letters and other characters; all tokens defined by the `\chardef` assignment; some tokens defined by `\let` or `\futurelet`.

3.3 Expanding tokens

The level 2 of TEX, i. e., the expansion level, reads tokens from the input token list and expands them. If the first token in the input token list is expandable, the level 2 of TEX expands it, that is, replaces this token (possibly with some tokens following it) with another sequence of tokens.

If—after the replacement—the first token is still expandable, the expansion is repeated until the list starts from an unexpandable token. Obviously, this process may loop infinitely.

For example, the result of expansion of the first token in the input token list:

$\boxed{\texttt{enspace}}$`Don...`

is the sequence of tokens:

$\boxed{\texttt{kern}}$`.5em␣Don...`

because the first token $\boxed{\texttt{enspace}}$ is expandable (it is a plain TEX macro) and its expansion is '`\kern.5em␣`'. The token $\boxed{\texttt{kern}}$ is unexpandable, hence no further expansion takes place.

The input token list with an unexpandable token at the beginning is submitted to the level 3 of TEX.

3.4 Commands

By a *command* we mean an unexpandable (primitive) token at the beginning of the input token list. If a command may or must have arguments, only the first token is a command. For example, in the input token list:

$\boxed{\texttt{kern}}$`.5em␣Don...`

the token $\boxed{\texttt{kern}}$ is the command and the tokens '`.5em␣`' are arguments. They are being read as a part of the process of executing the command.

In general, a command can read arguments from an input list either demanding expansion from the level 2 or not.

The level 3 of TEX—the level that executes commands—is the central level. Every time this level is about to execute the next command it 'asks' the level 2 to prepare the input token list such that at the beginning of the list there is a primitive (unexpandable) token. In turn, the level 2 'asks' the level 1 for preparing necessary tokens.

The level 3 executes the command according to its meaning, taking into account the current internal state of TEX, including the values of various parameters, and, in particular, taking into account current TEX's *mode*.

One of the results of executing commands is creation of various kinds of internal lists. The types of lists include: horizontal, vertical and math lists.

At every moment TEX is in one of the following six modes determining what type of list it is currently constructing:

(a) vertical mode (v-mode)
(b) internal vertical mode (iv-mode)
(c) horizontal mode (h-mode)
(d) restricted horizontal mode (rh-mode)
(e) math mode
(f) display math mode

At the very beginning of a job TEX is in v-mode and all the lists are empty. A list is constructed by appending new elements to it. The process of list construction can be briefly summarized as follows: mathematical lists are converted into h-lists; an h-list created in h-mode (a material for a paragraph) is converted into a v-list and appended to a current v-list; a vertical list created in v-mode is converted to boxes by a page builder; eventually, boxes to which a command `\shipout` is applied are written to a `DVI` file.

4 Summary of paragraph construction

In the process of creating a paragraph by TEX there are three distinct phases:

P1. Switching from v-mode to h-mode (opening a new h-list—see Section 5).
P2. Creating the h-list. (We do not discuss this phase in the paper. The notion of h-list is explained in 'The TEXbook,' pp. 94–95. The systematic description of how the commands processed in h-mode influence the state of the h-list contain chapters 24 and 25 of 'The TEXbook,' pp. 267–287).

P3. Switching from h-mode to v-mode (converting the h-list into a v-list and appending this vertical list to the main v-list; this is discussed in Section 7).

We will focus our attention on the moment of switching from v-mode or iv-mode to h-mode and back again. For the sake of simplicity we confine ourselves to the case where display math is not used inside a paragraph.

5 Switching from vertical to horizontal mode

In this section we describe *when* and *how* the level 3 of TEX accomplishes the change of modes from v-mode or iv-mode to h-mode.

First we say 'when', i. e., we list the commands that—if executed in one of v-modes—switch TEX's state to h-mode.

Then we say 'how', that is, we list the actions that TEX performs during the mode change.

5.1 Switching from vertical to horizontal mode: when

Some commands will be called here *vh-switches*, because if encountered in v-mode or in iv-mode they switch TEX to h-mode. They can be classified into two groups:

(a) explicite vh-switches:
- `*\indent`;
- `*\noindent`;

(b) implicit vh-switches (called by Knuth horizontal commands):
- letter: any character token of category 11 (also implicit; for example, control sequence `\d` after executing the assignment '`\let\d=A`'; the assignment associates the token `d` with a meaning that is primitive in TEX);
- other character: any character token of category 12 (also implicit; for example, control sequence `\one` after executing the assignment '`\let\one=1`');
- `*\char`;
- a 'chardef' token, i. e., a control sequence or an active character which has been assigned a meaning by the command `\chardef` (for example, control sequence `\ae` after the assignment '`\chardef\ae="1A`'; once again, the assignment associates the token `ae` with a meaning that is primitive in TEX);
- `*\noboundary` (a new primitive that appeared in TEX 3.0);
- `*\unhbox`, `*\unhcopy` (independently of the contents of the box being an argument);
- `*\valign`;
- `*\vrule`;
- `*\hskip`;
- `*\hfil`, `*\hfill`, `*\hss`, `*\hfilneg` (these tokens are primitive, not macros, even though the effects they cause could be achieved using `*\hskip` with appropriate parameters);
- `*\accent`;
- `*\discretionary`, `*\-`;
- `*\␣` (control space `*\␣` is a primitive command and if used in v-mode switches the mode to horizontal; note that normal space ␣, in general any space token, is ignored in v-mode);
- `$` (also the first `$` of the pair `$$` starting the displayed math formula).

It should be stressed that commands `*\hbox`, `*\vbox` and `*\vtop` are not switches. Such commands encountered in v-mode do not change the mode. The box (preceded by proper glue) is appended to the current v-list.

5.2 Switching from vertical to horizontal mode: how

Assume that TEX is in either v-mode or iv-mode. When the level 3 encounters a vh-switch at the beginning of the input token list it performs in turn the following actions:

(a) Optionally, a vertical glue `\parskip` is appended to the vertical list:
- if TEX is in iv-mode and the list is empty, the glue is not appended,
- if TEX is in iv-mode and the list is not empty, the glue is appended,
- if TEX is in v-mode the glue is always appended to the part called 'recent contributions' of the main v-list.

(b) If TEX is in v-mode (not iv-mode) the page builder is exercised, that is TEX runs the algorithm that moves elements of the v-list from the part of 'recent contributions' to the part 'current page'. In particular it may cause page breaking (running the `\output` routine).

(c) Switching from v-mode or iv-mode to h-mode occurs.

(d) Variables `\spacefactor` and `\prevgraf` are assigned values 1000 and 0, respectively (these assignments are called by Knuth 'global intimate assignments' and work in a rather peculiar way).

(e) A new h-list is initialised in the following way:
- if the vh-switch that caused the mode change was `*\noindent`, the newly created h-list is empty;
- if the vh-switch that caused the mode change was anything else (`*\indent` or any horizontal command), an empty box of width `*\parindent` is put at the beginning of the h-list.

(f) The following elements are appended to the beginning of the input token list:
- the contents of the token register `\everypar` (normally this register is empty),
- the vh-switch, provided it is a horizontal command; thus the explicit vh-switches `*\indent` and `*\noindent` are *not* put back into the input token list.

The rest of the input token list remains unchanged.

(g) Execution of the commands from the input token list starts. The commands are supplied by the level 2 of TEX.

6 Answer to the question Q1

Let us recall the question **Q1** of Section 2. We have asked about the difference between

(∗) `\everypar{\def\indent{1}}`
`\indent 3 is a prime number.`
and
(∗∗) `\everypar{\def\vrule{1}}`
`\vrule 3 is a prime number.`

From the point (f) of the list of actions performed by TEX at the beginning of a paragraph (Section 5.2) we can draw the following conclusions: if a paragraph has stared from the `\indent` command, the token `indent` is not put back to the input token list, therefore after executing the actions (a)–(f) the input token lists differ in both cases.

In the case (∗) the list is: '`def` `indent` `{1}3␣is␣a␣prime␣number.`'; in the case (∗∗) the list contains one more token: '`def` `vrule` `{1}` `vrule` `3␣is␣a␣prime␣number.`'.

Since redefining `indent` has nothing to do with the remainder of the list, the typesetting result in the case (∗) will be '3 is a prime number.'

In the case (∗∗) the token `vrule` is first defined as a macro expanding to the token `1` and then the newly defined macro `vrule` is expanded to `1`. Therefore in this case the result will be '13 is a prime number.'

This example shows some of consequences of the rule that the explicit vh-switches (`indent` and `noindent`) are not put back into the input token list after switching to h-mode.

7 Switching from horizontal to vertical mode

When the level 3 of TEX executes commands in h-mode, some commands cause closing the h-list and performing some actions that lead to switching from h-mode to v-mode.

In Section 7.1 below we say *when* TEX switches from h-mode to v-mode, i.e., we list the commands that cause switching. Then we explain *how* this mode change is performed.

7.1 Switching from horizontal to vertical mode: when

The commands listed below are called hv-switches, because if executed in h-mode they usually cause TEX to complete the h-mode and switch back to the enclosing v-mode or iv-mode. Similarly to the case of vh-switches, there are two groups of switches:

(a) explicit hv-switches:
- `*\par` (any token the current meaning of which is the same as the meaning of the token `par` when TEX starts job);

(b) implicit hv-switches (called by Knuth vertical commands):
- `*\unvbox`;
- `*\unvcopy`;
- `*\halign`;
- `*\hrule`;
- `*\vskip`;
- `*\vfil`;
- `*\vfill`;
- `*\vss`;
- `*\vfilneg`;
- `*\end`;
- `*\dump`.

7.2 Switching from horizontal to vertical mode: how

The behavior of TEX when it reads a hv-switch heavily depends on the type of the switch. If the switch is a vertical command (implicit hv-switch), TEX proceeds as follows:
- it inserts a token `par` at the beginning of the input token list (*before* the hv-switch token), regardless of the meaning of the `par` token;
- it starts executing commands from the input list (possibly expanding `par` if currently it is a macro).

It should be emphasized that TEX *does not change* the mode before reading the token `par` and that the expanded meaning of `par` may redefine the token that triggered the action (please note the danger of looping).

If the switch is explicit (`*\par`), TEX 'truly' finishes the paragraph, performing all or some of the actions (a)–(h) listed below.

TEX's behavior depends on whether the h-list is empty or not at the moment. If the h-list *contains at least one element*, all of the actions (a)–(h) are performed. If the h-list *is empty*, only the actions marked with an asterisk are executed, i. e., (e), (g) and (h).

All possible actions are:

(a) discarding the final element of the h-list, provided it is glue or leaders;

(b) appending to the end of the h-list the following three elements:

- `\penalty10000` (forbid break),
- glue of the size `\parfillskip`,
- `\penalty-10000` (force break);

(c) fixing the line-breaking parameters to be used in the next step,
(d) breaking h-list into lines and transforming this list into a v-list being the sequence of boxes, glue, penalty items and possibly other elements;
*(e) switching from h-mode back to the enclosing v-mode or iv-mode;
(f) appending the v-list created in step (d) to the enclosing v-list;
*(g) restoring the basic values of the parameters:
- `\parshape=0`, `\hangindent=0pt`, `\hangafter=1` (influencing the shape of a paragraph),
- `\looseness=0` (influencing the number of lines of a paragraph);

*(h) exercising the page builder if the current mode is the v-mode (but not iv-mode), i. e., initiating the process of moving elements from the recent contribution part of the vertical list to the current page.

8 Answer to the question Q2

The question was:

What is the difference between:
(∗) `\parindent=0mm \indent\par`
and
(∗∗) `\noindent\par`

Recall that we start in v-mode. The assignment of (∗) '`\parindent=0mm`' is just an assignment and does not append anything to the v-list. In both cases the command switching to h-mode (`\indent` or `\noindent`) causes appending the vertical glue of the size `\parskip` to the vertical list.

The command `\par` works differently in both cases (see Section 7.2) because h-lists constructed are different:
(∗) h-list at the moment of executing of the `\par` command contains a box of width 0 mm,
(∗∗) h-list at the moment of executing of the `\par` command is empty (the `\noindent` command does not append anything to the h-list).

So, according to what has been said in Section 7.2, points (a) and (b), in the case (∗) TEX 'breaks into lines' a list containing:
- the empty box,
- `\penalty10000`,
- `\parfilskip` glue,
- `\penalty-10000`.

The result is a one-line paragraph that is appended to the v-list as a single box preceded by a `\parskip` glue and an interline glue.

In the case (∗∗) only the `\parskip` glue is appended to the vertical list, since the h-list is empty at the time the `\par` command is executed.

8.1 Answer to the question Q3

We have asked what was the state of TEX after (∗) executing `\par` and after (∗∗) executing `{\par}`.

As we already know, TEX reacts to the command `*\par` performing the sequence of actions listed in Section 7.2. The results of most of the actions do not depend on the current level of grouping. However, the assignments mentioned in (g) are local within the current group.

Normally, at the end of each paragraph, TEX sets the values of `\parshape`, `\hangindent`, `\hangafter` and `\looseness` to 0, 0 pt, 1 and 0 respectively. But if a paragraph ends with `{\par}` instead of `\par` these values are assigned locally within the group surrounding `\par`. After closing the group TEX restores the values that the parameters had before the group started.

So, if the parameters mentioned above had standard values before `\par` or `{\par}`, their values do not change in both cases. If at least one of these parameters had a nonstandard value before `\par` or `{\par}`, executing just the `\par` command would result in restoring the standard value of this parameter, while in the case of `{\par}` the value of this parameter would be the same as before.

For example, by redefining `\par` as `{\endgraf}` and separating paragraphs with blank lines one can conveniently retain the same `\parshape` for several consecutive paragraphs.

9 Conclusions

We would like to emphasize that not the questions and answers mentioned in this this paper are important.

Our goal was to convince the reader that having a detailed (or, even better, formal) specification of TEX's mechanisms one could easily deduce the behavior of TEX in all situations.

We have described here a small fragment of TEX's machinery. Although the description is only partial and not fully precise, we believe that it makes a lot of misterious reactions of TEX understandable and straightforward.

10 Acknowledgements

Tomek Przechlewski and Piotr Pianowski: thank you.

LaTeX at WKAP in Dordrecht

Rob de Jeu

rob.dejeu@wkap.nl

Abstract

In this article, I will explain a couple of things about the use of LaTeX at Wolters Kluwer Academic Publishers (WKAP), a scientific publisher in Dordrecht. WKAP came into existence in the '80s after the merger between D. Reidel Publishing Company, Martinus Nijhoff and Dr W. Junk. Each year, WKAP publishes approximately 200 scientific journals and about 500 books (in reality, 200 scientific journal titles translates into some 1200 journal issues).

1 History

Towards the end of the '80s, WKAP increasingly had to deal with authors who, in addition to their paper manuscript, also sent along a diskette of the electronic file. Given that WKAP authors are spread out over the whole world, you can imagine the colourful collection of diskette formats and word processing systems which were used.

Specifically with the astronomy journals, it was noted that the quality of the paper manuscript (the 'hard copy'), was in some cases surprisingly good. Formulas and mathematical symbols weren't just added by hand in-between the text, but professionally typeset. This was reason enough to take a good look at the accompanying files. Without exception, these were typeset using TeX.

In 1989, a consultant was hired to initiate the people at WKAP in the wonderful world of TeX.

2 The choice for LaTeX

It was quickly apparent that LaTeX was more advantageous for use at WKAP than just plainTeX. The division between content and layout was very appealing to the publisher. More so, because it fits in nicely with WKAP's future plans to use SGML to store scientific articles in an electronic database.

In the meantime, a TeX configuration was installed at the in-house typesetter so that they could process manuscripts supplied in LaTeX themselves. In cooperation with the scientific editors of two of our journals, *Celestial Mechanics* and *Space Science Reviews*, the consultant wrote a style file matching WKAP's house style for these journals. The `article` style file served as starting point for this.

The majority of the copy for these journals was supplied in LaTeX and could easily be processed in-house without much difficulty. In the relevant journals, LaTeX was announced in the instructions to authors and it was explained how authors could get a hold of the special `kluwer` style file. A separate example- and instruction-file was also included with the style file. This is because the `kluwer` style file has an article 'opening environment' which differs from the LaTeX `article` style file.

The use of LaTeX together with this style file was very successful. The number of articles delivered in LaTeX increased significantly. And, more often, the in-house typesetter opted to use LaTeX when typesetting traditional manuscripts.

3 The present situation

Right now, the situation at WKAP is as follows: A journal-independent style file has been written, `kluwer.sty`, and it serves as basis for 10 journal specific style files. In these journal specific style files, the `kluwer.sty` data is `\input`-ted and after that, the journal specific characteristics are arranged. The `kluwer` style file gives a 'sober' general layout in the Computer Modern Font and is used externally by our authors and editors. This way, we only have to support one external style file. This style file includes an example- and instruction-file which helps the author to use the style file. This is mainly for things which differ from the `article` style file, plus a couple of *do's* and *don'ts*.

After an article coded according to the `kluwer` style file is received, the article is printed out in the appropriate journal specific layout so as to generate a first proof. Likewise, the font is changed from Computer Modern to Times, and the so-called creditlines are added, running headlines adapted, and eventual corrections are made, etc. This takes place at the in-house setter or at one of the external typesetters who have converted to LaTeX in the meantime. Presently, around 50 journals are produced using LaTeX and for these journals, LaTeX files are accepted from authors. It's expected that this number will continue to increase significantly in the coming years.

4 The relationship between LaTeX and SGML

As mentioned above, one of the reasons for choosing LaTeX was the LaTeX-concept of a division between (structured) content and layout because it fits in nicely with SGML. At this moment, WKAP is busy making preparations to store scientific articles in an electronic database in SGML format. In order to generate paper products, LaTeX will be used as the setting system. Presently, different SGML DTDs[1] for scientific articles are being evaluated. When a DTD has definitively been chosen, then the `kluwer` style file will undoubtedly require a thorough revision so as to dove-tale nicely. For the different SGML elements in the DTD, macros can be written in the style file.

5 Conclusion

By now, it should be clear that LaTeX plays an important role at WKAP. On one hand, we are confronted with a growing number of manuscripts which are supplied in LaTeX. On the other hand, LaTeX is increasingly chosen as typesetting system for scientific articles and books. Soon, the availability of style files on a ftp server can only lead to an increase in the number of authors who supply their articles in LaTeX.

Above all, in the near future, LaTeX will be implemented as a typesetting system for generating paper products from the SGML files in the electronic database. Thus, the importance of LaTeX for WKAP can only but increase in the coming years!

[1] DTD: Document Type Definition. Abstract model of a document-type in which it's structure is described.

TEX en braille. . . een illusie?

Philippe Vanoverbeke

Langenhoekstraat 21B,
B-8210 Veldegem
+32 50 279853

Abstract

Ongetwijfeld zal bovenstaande titel menig NTG-lid de wenkbrauwen doen fronsen. . . . Typografie, laat staan TEX, is nu wel het laatste wat wij in verband brengen met onze blinde medemens. In de volgende paragrafen zal ik pogen te verklaren waarom ik TEX als een mogelijk hulpmiddel zie voor de produktie van (eenvoudige) documenten in braille.

1 Inleiding

Braille is, zoals eenieder wel weet, de manier waarop blinden in geschreven taal kunnen communiceren. De meest verspreide vorm is het 6-punts brailleschrift[1] waarmee 63 combinaties mee kunnen gevormd worden. Om dit verwezenlijken bestaat tal van apparatuur: van braille schrijfmachines tot gesofistikeerde braille printers. Vooral deze laatste hebben een prijsniveau dat nogal aan de hoge kant ligt (vanaf om en bij de 4.000 dollar) en dan hoort daar natuurlijk nog de nodige software bij die, hoe kan het anders, evenmin goedkoop te noemen is.

2 Waarom TEX ?

Naast de echte printers ('embosser') bestaat er ook een ander procédé, namelijk de stereo-copie of thermoform machine. Hiermee wordt een soort fotocopie van een blad genomen waarbij de zwarte 'tekening' omgezet wordt in reliëf. Dergelijke machines worden dus gebruikt om brailleteksten en tekeningen te copiëren. Tot voor kort waren deze machines evenzo onbetaalbaar maar nu is hun prijsniveau gezakt tot zowat 2.000 dollar voor een degelijk exemplaar.

Tot zover niets bijzonder, ware het niet dat voor TEX reeds braille-fonts ontworpen zijn (door Dr. Wolfgang Leister van het 'Institut für Betriebs- und Dialogsysteme' in Karlsruhe, Duitsland). Dankzij Gerard Van Nes kwam ik in het bezit van deze fonts en kon ik beginnen 'experimenteren'. Vermits ik in België slecht één thermoform machine kon vinden (blindeninstituut Spermalie te Brugge) en de bladprijs mij even tot inkeer bracht (4 gulden per blad. . .) hield ik het even voor bekeken tot ik in juni j.l. een fabrikant vond die dergelijke machines (en supplies) met normale prijzen op de markt brengt.

Punt is dat ik nu — zo dacht ik eerst — over een goedkoop alternatief zou kunnen beschikken om teksten in braille te 'zetten' (zwart-wit hardcopy door TEX en duplicatie hiervan op een thermoform machine). Vermits ik een blind zoontje van 7 heb en nogal wat teksten produceer met TEX (lees: LATEX) leek mij dit een goede en vooral voordelige oplossing.

3 De fonts. . .

Zoals vermeld werden er reeds fonts aangemaakt door Dr. W. Leister. Meteen bleek echter dat deze gebaseerd zijn op het Duitse blindengeschrift. Inderdaad, braille kent verschillende implementaties volgens het taalgebied. Afhankelijk van de taal wordt in braille één braille-teken gebruikt voor veel voorkomende letterkombinaties. In het Nederlands betreft dit de letterkombinaties ch, sch en oe, voor een andere taal is dat natuurlijk anders. . . . Tevens wordt bij deze Duitse versie voor hoofdletters 8-punts braille gebruikt, terwijl er in de Nederlandstalige 6-punts braille een extra 6-punts braille-teken gebruikt wordt.

De momenteel beschikbare fonts zijn dus voor ons taalgebied (Nederland en België) niet geschikt. Ondergetekende is dus druk bezig om zich te verdiepen in Metafont om binnen afzienbare tijd — althans dat hoop ik — een bruikbare versie van deze fonts te verwezenlijken. Sinds deze zomer verblijf ik echter regelmatig op zee wat niet bepaald bevorderlijk is voor mijn plannen terzake.

4 De voor- en nadelen op een rij

Er zijn volgens mij genoeg redenen om LATEX of TEX aan te wenden als braille productiemiddel:

1. De combinatie van TEX voor de aanmaak van een zwart-wit copie met een thermoform machine kan een besparing vormen van enkele duizenden tot tienduizenden guldens. Hierdoor zou de productie van braille documenten tot een betaalbaar niveau teruggebracht kunnen worden, niet alleen voor par-

[1] Daarnaast bestaat het 8-punts brailleschrift bedoeld voor wetenschappelijke documenten.

ticulieren maar ook voor instellingen.
2. TEX en LATEX zijn intrinsiek superieur aan de momenteel gebruikte tekstverwerkers welke gebruikt worden voor de aanmaak van braille (in het 'beste geval' WP), zeker wanneer het grotere documenten betreft;
3. Previewen van een braille document is mogelijk (niet bij alle andere systemen mogelijk);
4. Geen dure specifieke apparatuur en/of dure insteekkaarten zijn benodigd, een eenvoudige PC met printer en thermoform-machine volstaan (5 à 6.000 gulden);
5. Met uitzondering van de thermoform-machine is de apparatuur ook voor andere doeleinden geschikt.

Er zijn ook nadelen:
1. Een dergelijk productieproces is enkel praktisch bruikbaar door 'zienden';
2. Er is enige – zij beperkte – TEX kennis vereist;

5 Hoe moet het nu verder?

Praktisch gezien is TEX en/of LATEX nog steeds niet bruikbaar gezien de onvolkomendheden van de fonts voor ons taalgebied. Ik heb mij dan ook tot doel gesteld om binnen afzienbare tijd m.b.v. Metafont de benodigde fonts aan te maken voor 6-punts braille. Advies van TEX goeroes is natuurlijk altijd welkom. . .

Eens dit tot een goed einde gebracht dient een aangepaste versie voor 8-punts braille op punt gesteld en de bijbehorende styles bijgeschaafd.

6 Besluit

Hoewel TEX zeker nooit bedoeld is geweest om braille te produceren biedt dit systeem toch de mogelijkheid om, in combinatie met een thermoform machine, op een soepele en goedkope manier gewone tekst in braille om te zetten.

Op de huidige markt bestaan reeds diverse systemen die braille kunnen aanmaken. Het prijsniveau van deze producten is echter van die aard dat zelfs instellingen ze helemaal niet of in te beperkte hoeveelheid kunnen aanschaffen (ik vernoem met opzet geen merken, noch instellingen). Dit is een situatie die, naar ik mag verhopen, zal verbeteren naarmate het informatica-materiaal goedkoper wordt.

Wat mij persoonlijk betreft: liever een werkend systeem op basis van TEX in de hand dan 10 onbetaalbare systemen in de lucht.. . .

7 Voorbeeld

Op de volgende pagina is een voorbeeld in braille opgenomen. De tekst doet er overigens niet toe, doel is de lezer een idee te geven wat een pagina in braille voorstelt. Op te merken valt dat een tekst in braille veel omvangrijker is in pagina's dan de originele tekst. Dit is een extra reden om de bladprijs bij de produktie zo laag mogelijk te houden.

```
\documentstyle[braille,a4wide]{article}
\begin{document}
 \begin{braille}
  Geborgenheid is vooral tijdens de
  eerste levensjaren van fundamenteel
  belang. Het levert immers een
  basisveiligheid en bescherming op
  voor het kind. Dit betekent niet
  dat het kind hierdoor beschermd is
  tegen de obstakels en de
  moeilijkheden die zijn leven
  mogelijks kunnen doorkruisen.

  De basisveiligheid die het kind
  vanuit de geborgenheid verwerft,
  biedt wel de ruimte om de
  omringende wereld te exploreren.
 \end{braille}
\end{document}
```