

Fonts in Context

Hans Hagen
October 2001

Contents

1	Introduction	1
2	Font files and encodings	2
3	Simple font definitions	2
4	Defining body fonts	3
5	Typescripts and typefaces	6
6	Spacing	15
7	Encodings and mappings	16
8	Regimes	22
9	Font handling	22
10	Math collection	28
11	Predefined typefaces	30
12	Symbols and glyphs	31
13	Map files	32
14	Installing fonts	32
15	Getting started	36

1 Introduction

This manual is no replacement for the reference manual but an addendum. Here we will cover some details of defining fonts and collections of fonts, called typefaces. We will also spend some words on installing fonts. In any case, it helps if you know what a font is, and are familiar with the `CONTEXT` font switching macros.

The original `CONTEXT` font model was based on plain `TEX`, but evolved into a more extensive one primarily aimed at consistently typesetting our educational documents. The fact that we had to typeset pseudo caps in any font shape in normal text as well as superscript mode, has clearly determined the design. This model has been relatively stable since 1995.

Currently there are three layers of font definitions:

- simple font definitions: such definitions provides `\named` access to a specific font in a predefined size
- body font definitions: these result in a coherent set of fonts (often) from a same type foundry (or designer) that can be used intermixed
- typeface definitions: they package serif, sans serif, mono spaced and math and other styles in such a way that you can conveniently switch between different combinations

These three mechanisms are actually build on top of each other and all fall back on a low level mapping mechanism that is responsible for resolving the real font file name and the specific font encoding used.

When T_EX users install one of the T_EX distributions, like T_EX-live, they will have a lot of fonts already on the system. Unfortunately it is not that easy to get a clear picture of what is there and what is needed to use them. Although the `texmf` tree is prepared for commercial fonts, adding them is not trivial. To compensate this, CON_TE_XT comes with `texfont.pl`, that can install fonts for you. Also, too help you on way, we provide typescripts for a couple of free fonts.

2 Font files and encodings

In CON_TE_XT when possible you should use symbolic names for fonts. The mapping from these names onto real ones in most cases goes unnoticed for the user. This is good since the name depends on the encoding and therefore not seldom is obscure and hard to remember.

```
\definefontsynonym [Serif]      [Palatino]
\definefontsynonym [Palatino] [uplr8t]  [encoding=ec]
```

The advantage of using for instance `Serif` in definitions is, that we can later easily remap this name onto another font than `Palatino`. In a similar way, we can define new names that map onto `Serif`.

```
\definefontsynonym [TitleFont] [Serif]
```

By using symbolic names in for instance style and macro definitions, you can make them independent of a particular font and let themselves adapt to the main document fonts, which normally are defined in terms of `Serif`.

There is no limitation on the level of mapping, but the last one in the chain has to be a valid font filename. Specific font encoding declarations take place at that level, since they are closely related to specific instances of fonts. We come back to this in later sections.

3 Simple font definitions

The most simple font definition takes place with `\definefont`. If you want a fixed size, you can define a font as follows:

```
\definefont [TitleFont] [Serif at 24pt]
```

The `at` specifier is a natural \TeX one, just as `scaled`. But where `at` is useful, `scaled` is rather useless, since it scales the font related to its design size which is often unknown. Depending on the design size is especially dangerous when you use symbolic names, since different fonts have different design sizes, and designers differ in their ideas about what a design size is. Compare for instance the 10pt instance of a Computer Modern Roman with Lucida Bright (which more looks like a 12pt then).

```
\definefont [TitleFont] [Serif scaled 2400]
```

Hard codes sizes can be annoying when you want to define fonts in such a way that their definitions adapt themselves. Therefore we provide an additional way of scaling:

```
\definefont [TitleFont] [Serif sa 2.4]
```

The `sa` directive means as much as ‘scaled at the body font size’. Therefore this definition will lead to a 24pt scaling when the (document) body font size equals 10pt. Because the definition has a lazy nature, the font size will adapt itself to the current body font size.

Instead of a number, you can also use an identifier, as defined in the body font environment that specifies related dimensions. This scales the font to the `b` size, being 1.440 by default.

```
\definefont [TitleFont] [Serif sa b]
```

An alternative to `sa` is `mo`. Here the size maps onto the remapped body font size when given. We will not cover this in detail here.

4 Defining body fonts

The core of this model is the definition command that is used as follows:

```
\definebodyfont [10pt] [rm] [tf=tir at 10pt]
```

As one can expect, the first implementation of a font model in \TeX is also determined and thereby complicated by the fact that the Computer Modern Roman fonts come in design sizes. As a result, definitions can look rather complex and because most \TeX users start with those fonts, font definitions are considered to be complex.

Another complicating factor is that in order to typeset math, even more definitions are needed. Add to that the fact that sometimes we need to use fonts with mixed encodings, i.e. with the glyphs positioned in different font slots, and you can understand

why font handling in \TeX is often qualified as ‘the font mess’. Flexibility simply has its price.

Many documents have a rather simple design and use only a couple of (often related) fonts. For some commonly used fonts, this means that one can stick to loading the appropriate predefined font definition file.¹ But font life is seldom simple and, in a more worst case scenario, one must define the fonts in the document style.

Because most fonts come in one design size, we can simplify the definitions by using predefined sizes, like the default one (type sa 1):

```
\definebodyfont [10pt,11pt,12pt] [rm] [default]
```

The default relations between sizes are determined by the body font environment. You can get some insight in this by typesetting this environment as shown in figure 1.

```
\showbodyfontenvironment % [1br]
```

[11.0pt]							
text	script	scriptscript	x	xx	small	big	interlinie
20.7pt	14.4pt	12pt	17.3pt	14.4pt	17.3pt	20.7pt	
17.3pt	12pt	10pt	14.4pt	12pt	14.4pt	20.7pt	
14.4pt	11pt	9pt	12pt	10pt	12pt	17.3pt	
12pt	9pt	7pt	10pt	8pt	10pt	14.4pt	
11pt	8pt	6pt	9pt	7pt	9pt	12pt	
10pt	7pt	5pt	8pt	6pt	8pt	12pt	
9pt	7pt	5pt	7pt	5pt	7pt	11pt	
8pt	6pt	5pt	6pt	5pt	6pt	10pt	
7pt	6pt	5pt	6pt	5pt	5pt	9pt	
6pt	5pt	5pt	5pt	5pt	5pt	8pt	
5pt	5pt	5pt	5pt	5pt	5pt	7pt	
4pt	4pt	4pt	4pt	4pt	4pt	6pt	

Figure 1 The current bodyfont environment.

Because the font names (may) depend on the encoding vector, we can use the previously discussed method for mapping symbolic names. So, one can comfortably say:

The original font definition files are replaced by typescripts in `type-pre`, but font files are still supported
¹ for upward compatibility reasons.

```
\definebodyfont [10pt,11pt,12pt] [rm] [tf=tir          sa 1]
\definebodyfont [10pt,11pt,12pt] [rm] [tf=Times-Roman sa 1]
\definebodyfont [10pt,11pt,12pt] [rm] [tf=Serif        sa 1]
```

As we already pointed out, the mapping from symbolic names onto the real file name can be direct or indirect. The indirect method has the advantage that one can also use the more abstract name (*Serif*) as well as the real name (*Times-Roman*), but can leave the file name untouched. Document styles thereby can be defined in such a way that they are independent of font file names. This means that the previous definition can become:

```
\definebodyfont    [10pt,11pt,12pt] [rm] [tf=Serif sa 1]
\definefontsynonym [Serif] [Times-Roman]
\definefontsynonym [Times-Roman] [tir] [encoding=texnansi]
```

These commands permit you to combine fonts in any way in any size, but when documents have a more complicated design, there may be many *Serif*'s and multiple math fonts used. Of course this can be handled, but only by redefining fonts at the spot and this is not only cumbersome, but also undesirable from the perspective of document source management.

Consider the following text:

```
Who is {\it fond} of fonts?
Who claims that $t+e+x+t=m+a+t+h$?
Who {\ss can see} {\tt the difference} here?
```

In Computer Modern Roman fonts, this looks like:

```
Who is fond of fonts?
Who claims that  $t + e + x + t = m + a + t + h$ ?
Who can see the difference here?
```

While in Lucida it shows up as:

```
Who is fond of fonts?
Who claims that  $t + e + x + t = m + a + t + h$ ?
Who can see the difference here?
```

The standard POSTSCRIPT font have yet another look and feel:

```
Who is fond of fonts?
Who claims that  $t + e + x + t = m + a + t + h$ ?
Who can see the difference here?
```

As you can notice, there are differences in size and shape. The switch between those fonts was done by issuing the following commands.

```
\switchtobodyfont[cmr]
\switchtobodyfont[lbr]
\switchtobodyfont[pos,tim]
```

With `\showbodyfont[...]` we can get a summary of such a font collection.

```
\showbodyfont[cmr]
\showbodyfont[lbr]
\showbodyfont[pos]
```

[cmr]										\mr : Ag			
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	AG	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Figure 2 Computer Modern Roman.

[lbr]										\mr : Ag			
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	AG	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Figure 3 Lucida Bright.

This way of switching fonts has been part of `CONTEX`T from the beginning, but as more complicated designs started to show up, we felt the need for a more versatile mechanism.

5 Typescripts and typefaces

On top of the existing (but extended) traditional font module, we now provide a more abstract layer of typescripts and building blocks for definitions and typefaces as font

[pos]												\mr : Ag	
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Figure 4 Times Roman, Helvetica & Courier.

containers. The original font definition files have been regrouped into such typescripts thereby reducing the number of files involved.

Typescripts are in fact just organized definitions. The previously shown Lucida Bright font collection, can be defined as follows. First we map some symbolic names onto Lucida names; the mapping to encoding specific filenames takes place somewhere else.

```

\definefontsynonym [Serif]           [LucidaBright]
\definefontsynonym [SerifBold]       [LucidaBright-Demi]
....
\definefontsynonym [Sans]            [LucidaSans]
\definefontsynonym [SansBold]        [LucidaSans-Demi]
....
\definefontsynonym [Mono]            [LucidaSans-Typewriter]
\definefontsynonym [MonoBold]        [LucidaSans-TypewriterBold]
....
\definefontsynonym [MathRoman]       [LucidaBright]
\definefontsynonym [MathExtension]   [LucidaNewMath-Extension]

```

Because no design sizes are involved, we can define the sizes in a rather fast way.

```

\definebodyfont
  [17.3pt,14.4pt,12pt,11pt,10pt,9pt,8pt,7pt,6pt,5pt,4pt]
  [rm,ss,tt,mm]
  [default]

```

As you can see here, these definitions define the `serif`, `sans`, `mono` and `math` shapes together. In the typescript layer, these definitions are split:

```

\starttypescript [serif] [lucida] [name]
  \definefontsynonym [Serif]       [LucidaBright]

```



```

\definefontsynonym [SerifBold] [LucidaBright-Demi]
....
\stoptypescript

```

In a similar way the sizes have become typescripts:

```

\starttypescript [serif] [default] [size]
  \definebodyfont
    [17.3pt,14.4pt,12pt,11pt,10pt,9pt,8pt,7pt,6pt,5pt,4pt]
    [rm] [default]
\stoptypescript

```

The definition of the Lucida Bright font collection can now be simplified to:

```

\starttypescript [lbr]
  \usetypescript [all] [lucida] [name]
  \usetypescript [all] [default] [size]
\stoptypescript

```

Typescripts and its invocations have upto three specifiers. An invocation matches the script specification when the three arguments have common keywords. The special keyword `all` is equivalent to any match. Although any keyword is permitted, the current definitions have some reserved (advised) keys, like:

pattern	application
[serif] [*] [*]	serif fonts
[sans] [*] [*]	sans serif fonts
[mono] [*] [*]	mono spaced fonts
[math] [*] [*]	math fonts
[*] [*] [size]	size specification
[*] [*] [name]	symbolic name mapping
[*] [*] [special]	special settings
[*] [*] [special]	special settings
[*] [default] [*]	default case

In many cases the font class or encoding is part of the specification. These are variable.

pattern	application
[*] [class] [*]	a specific font class
[*] [*] [encoding]	a specific font encoding

When you take a close look at the files you will notice a couple of more keywords, but we will not discuss them here. Instead of the predefined size `default`, you can use the `dtp` size scripts with their associated body font environments.

In the example of the Lucida Bright definition, we still treat the font as a whole: `serif`, `sans`, `mono` and `math` come from one family of fonts. Instead of defining the font this way, we could have created a so called typeface collection. Such a definition looks as follows:

```
\definetypeface [funny] [rm]
  [serif] [lucida] [default] [encoding=texnansi]
\definetypeface [funny] [ss]
  [sans] [lucida] [default] [encoding=texnansi]
\definetypeface [funny] [tt]
  [mono] [lucida] [default] [encoding=texnansi]
\definetypeface [funny] [mm]
  [math] [lucida] [default] [encoding=texnansi]
```

From this moment, `\funny` will enable this specific collection of fonts. In a similar way we can define a collection `\joke`.

```
\definetypeface [joke] [rm]
  [serif] [times] [default] [encoding=texnansi]
\definetypeface [joke] [ss]
  [sans] [helvetica] [default] [rscale=.9,encoding=texnansi]
\definetypeface [joke] [tt]
  [mono] [courier] [default] [rscale=1.1,encoding=texnansi]
\definetypeface [joke] [mm]
  [math] [informal] [default] [encoding=texnansi]
```

And the familiar Computer Modern Roman as `\whow`:

```
\definetypeface [whow] [rm]
  [serif] [computer-modern] [computer-modern] [encoding=ec]
\definetypeface [whow] [ss]
  [sans] [computer-modern] [computer-modern] [encoding=ec]
\definetypeface [whow] [tt]
  [mono] [computer-modern] [computer-modern] [encoding=ec]
\definetypeface [whow] [mm]
  [math] [computer-modern] [computer-modern] [encoding=ec]
```

When typeset in `\funny`, `\joke`, and `whow`, the samples now look like:

Who is *fond* of fonts?

Who claims that $t + e + x + t = m + a + t + h$?

Who can see the difference here?

Who is *fond* of fonts?

Who claims that $t + e + x + t = m + a + t + h$?

Who can see **the** difference here?

Who is *fond* of fonts?

Who claims that $t + e + x + t = m + a + t + h$?

Who can see the difference here?

With `\showbodyfont` you can get an overview of this font.

[funny]										\mr : Ag			
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Figure 5 The *funny* typeface collection.

[joke]										\mr : Ag			
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Figure 6 The *joke* typeface collection.

When defining the joke typeface collection, we used a scale directive. The next sample demonstrates the difference between the non scaled and the scaled alternatives.

Who is *fond* of fonts?

Who claims that $t + e + x + t = m + a + t + h$?

Who can see the difference here?

Who is *fond* of fonts?

Who claims that $t + e + x + t = m + a + t + h$?

[whow]										\mr : Ag			
	\tf	\sc	\sl	\it	\bf	\bs	\bi	\tfx	\tfxx	\tfa	\tfb	\tfc	\tfd
\rm	Ag	AG	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\ss	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag
\tt	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag	Ag

Figure 7 The `whow` typeface collection.

Who can see the difference here?

In due time `CONTEXT` will come with more predefined typeface collections. One of the currently predefined typefaces is Computer Modern Roman:

```
\usetypscript[modern][ec]    % for western european users
\usetypscript[modern][il2]   % for czech and slovak users
\usetypscript[modern][p10]   % for polish users
```

Another set is made up by the Adobe's standard 15 fonts:

```
\usetypscript[postscript][texansi] % our preferred encoding
\usetypscript[postscript][ec]      % another popular one
```

It may not be clear from the previous examples, but a big difference between using typeface definitions and the old method of redefining over and over again, is that the new method uses more resources. This is because each typeface gets its own name space assigned. As an intentional side effect, the symbolic names also follow the typeface. This means that for instance:

```
\definefont[MyBigFont][Serif sa 1.5] \MyBigFont A bit larger!
```

will adapt itself to the currently activated serif font shape, here `\funny`, `\joke` and `\whow`.

A bit larger!
A bit larger!
A bit larger!

The option to define relative font sizes using the `rscale` parameter permits fine tuning of font sizes. Fine tuning of the sizes `x`, `xx`, `a`, `b`, ... as well as interline spacing is handled by the `bodyfont` environment. This command normally takes two arguments,

but accepts an optional (first) one denoting a class. You can use this command to tailor the environment for a specific typeface.

Although the default interline space is quite well tuned to the average font, you may want to change it using this command. The defaults used to typeset this paragraph are related to the x-height of the font.

```
\definebodyfontenvironment
[joke] [11pt]
[interlinespace=20pt]
```

However, keep in mind that when you change the dimensions for one size, you also need to change them for other sizes in order to get a consistent look and feel when switching to a smaller or larger size.

Math is kind of special in the sense that it has its own set of fonts, either or not related to the main text font. By default, a change in style, for instance bold, is applied to text only.

```
$      \sqrt{625} =      5\alpha$
$\bf   \sqrt{625} =      5\alpha$
$      \sqrt{625} = \bf 5\alpha$
$\bfmath \sqrt{625} =      5\alpha$
```

The difference between these four lines is as follows:

```
\sqrt{625} = 5\alpha
\sqrt{625} = 5\alpha
\sqrt{625} = 5\alpha
\sqrt{625} = 5\alpha
```

In order to get a bold α symbol, we need to define bold math fonts.² The most convenient way of doing this is the following:

```
\definetypeface [funny] [mm]
[math,boldmath] [lucida] [default] [encoding=texnansi]
```

Bold math looks like this:

```
\sqrt{625} = 5\alpha
\sqrt{625} = 5\alpha
\sqrt{625} = 5\alpha
\sqrt{625} = 5\alpha
```

Bold math is already prepared in the core modules, so normally one can do with less code

The definitions are given on the next page. Such definitions are normally collected in the project bound file, for instance called `typeface.tex`. You can add a filename to the list of typescript files yourself:

```
\usetypescriptfile[typeface] % project scripts
```

An example of such a file is shown below:

```
% Additional user typescripts.

% First, we need to define the symbolic names for the new
% fonts. Because no script specification is given, it is only
% expanded once. This prevents unwanted overloading when the
% file is loaded more than once.

\starttypescript
  \definefontsynonym [MathRomanBold]      [MathRoman]
  \definefontsynonym [MathExtensionBold]  [MathExtension]
  \definefontsynonym [MathItalicBold]     [MathItalic]
  \definefontsynonym [MathSymbolBold]     [MathSymbol]
  \definefontsynonym [MathAlphaBold]      [MathAlpha]
  \definefontsynonym [MathBetaBold]       [MathBeta]
  \definefontsynonym [MathGammaBold]      [MathGamma]
  \definefontsynonym [MathDeltaBold]      [MathDelta]
\stoptypescript

% We define a new class ‘boldface’ and populate it with the
% Lucida fonts. The mapping onto real file names is handled
% in the encoding scripts.

\starttypescript [boldmath] [lucida] [name]
  \definefontsynonym [MathRomanBold]      [LucidaBright-Demi]
  \definefontsynonym [MathExtensionBold]   [LucidaNewMath-Extension]
  \definefontsynonym [MathItalicBold]      [LucidaNewMath-AltDemiItalic]
  \definefontsynonym [MathSymbolBold]      [LucidaNewMath-Symbol-Demi]
  \definefontsynonym [MathAlphaBold]       [LucidaNewMath-Arrows-Demi]
\stoptypescript

% We have to tell ConTeXt how the bold math fonts are scaled.

\starttypescript
  \definebodyfont
    [boldmath] [mm]
    [mrbf=MathRomanBold      mo 1,
     exbf=MathExtensionBold  mo 1,
     mibf=MathItalicBold     mo 1,
     sybf=MathSymbolBold     mo 1,
     mabf=MathAlphaBold      mo 1,
```

```

        mbbf=MathBetaBold      mo 1]
\stoptypescript

% This script is responsible for teh real definition of the
% bold math fonts. It will use the previously defined default
% values.

\starttypescript [boldmath] [default] [size]
\definebodyfont
    [17.3pt,14.4pt,12pt,11pt,10pt,9pt,8pt,7pt,6pt,5pt,4pt]
    [mm] [boldmath]
\stoptypescript

```

It is also possible to avoid typescripts. When definitions are used only once, it makes sense to use a more direct method. We will illustrate this with a bit strange example.

Imagine that you want some math formulas to stand out, but that you don't have bold fonts. In that case you can for instance scale them. A rather direct method is the following.

```

\definebodyfont
    [funny]
    [12pt,11pt,10pt,9pt,8pt,7pt] [mm]
    [mrbf=MathRoman      mo 2,
     exbf=MathExtension mo 2,
     mibf=MathItalic     mo 2,
     sybf=MathSymbol     mo 2]

```

Our math sample will now look like:

$$\begin{array}{l}
 \sqrt{625} = 5\alpha \\
 \sqrt{\mathbf{625}} = 5\alpha \\
 \sqrt{\mathbf{625}} = 5\alpha \\
 \sqrt{\mathbf{625}} = 5\alpha
 \end{array}$$

We can also use an indirect method:

```

\definebodyfont
    [smallmath] [mm]
    [mrbf=MathRoman      mo .5,
     exbf=MathExtension mo .5,
     mibf=MathItalic     mo .5,
     sybf=MathSymbol     mo .5]

```

```
\definebodyfont
  [funny]
  [12pt,11pt,10pt,9pt,8pt,7pt]
  [mm] [smallmath]
```

This method is to be preferred when we have to define more typefaces since it saves keystrokes.

```
 $\sqrt{625} = 5\alpha$ 
 $\sqrt{\mathbf{625}} = 5\alpha$ 
 $\sqrt{6\overline{25}} = 5\alpha$ 
 $\sqrt{62\overline{5}} = 5\alpha$ 
```

For efficiency reasons, the font definitions (when part of a typeface) are frozen the first time they are used. Until that moment definitions will adapt themselves to changes in for instance scaling and (mapped) names. Freezing definitions is normally no problem because typefaces are defined for a whole document and one can easily define more instances. When you redefine it, a frozen font is automatically unfrozen.

6 Spacing

The baseline distance as well as a couple of other spacing values are derived from the body font size. The main spacing is set up in such a way that it adapts itself to the current font size.

```
\setupinterlinespace[line=2.8ex]
```

Hard coded values (like 15pt) are kind of dangerous here since these inhibit `CONTEXT` to adapt itself. This command has some more parameters that are discussed in the reference manual. Here we limit the discussion to definitions of fonts.

Occasionally you may want to adapt the baseline distance (interline spacing) to a specific font, for instance a big title font on the cover. The best way to do this is:

```
\definefont [PiFont] [Serif sa 3.1415]
\PiFont \setupinterlinespace
```

or:

```
\definedfont [EFont] [Sans sa 2.71] \setupinterlinespace
```

If you do this grouped, you should end the paragraph inside the group, otherwise the spacing dimensions are forgotten:


```
{\PiFont \setupinterlinespace Fonts in \par \ConTeXt \par}
```

Instead of setting the spacing at the document level, i.e. for each font, you can set the spacing per body font environment:

```
\setupbodyfontenvironment
  [modern] [12pt]
  [interlinespace=14pt]
```

7 Encodings and mappings

Not every language uses the (western) latin alphabet. Although in most languages the basic 26 characters are somehow used, they can be combine with a broad range of accents placed in any place.

In order to get a character representation, also called glyph, in the resulting output, you have to encode it in the input. This is no problem for `a..z`, but other characters are accessed by name, for instance `\eacute`. The glyph é can be present in the font but when it's not there, T_EX has to compose the character from a letter e and an accent `.

In practice this means that the meaning of `\eacute` depends on the font and font encoding used. There are many of such encodings, each suited for a subset of languages.

encoding	usage
default	the 7 bit ASCII encoding as used by plain T _E X
texnansi	a combination of T _E X and Adobe standard encoding
8r	a (strange) mixture of encodings
ec	a rather complete encoding suitable for most languages
il2	iso latin 2 encoding as needed for Czech and Slovak
p10	a native Polish encoding

These encodings are font related as is demonstrated in figure 8, 9 and 10. Here we used the `\showfont` command.

The situation is even more complicated than it looks, since the font may be virtual, that is, build from several fonts.

The advantage of using specific encodings is that you can let T_EX hyphenate words in the appropriate way. The hyphenation patterns are applied to the internal data structures that represent the sequence of glyphs. In spite of what you may expect, they are font dependent! Even more confusing: they not only depend on the font

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
000	001	002	003	004	005	006	007	008	009	00a	00b	00c	00d	00e	00f
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
020	021	022	023	024	025	026	027	028	029	030	031	032	033	034	035
32	33	34	35	36	37	38	39	40	41	42	43	44	hyph	45	46
040	041	042	043	044	045	046	047	048	049	050	051	052	2a	053	054
48	49	50	51	52	53	54	55	56	57	58	59	60	2c	055	056
060	061	062	063	064	065	066	067	068	069	070	071	072	3a	073	074
64	65	66	67	68	69	70	71	72	73	74	75	76	3b	075	076
080	081	082	083	084	085	086	087	088	089	090	091	092	4c	115	093
100	101	102	103	104	105	106	107	108	109	110	111	112	4a	113	114
80	81	82	83	84	85	86	87	88	89	90	91	92	4b	114	115
120	121	122	123	124	125	126	127	128	129	130	131	132	5a	133	134
96	97	98	99	100	101	102	103	104	105	106	107	108	5b	135	136
140	141	142	143	144	145	146	147	148	149	150	151	152	6a	153	154
112	113	114	115	116	117	118	119	120	121	122	123	124	6b	155	156
160	161	162	163	164	165	166	167	168	169	170	171	172	7a	173	174
128	129	130	131	132	133	134	135	136	137	138	139	140	7b	175	176
200	201	202	203	204	205	206	207	208	209	210	211	212	8a	213	214
144	145	146	147	148	149	150	151	152	153	154	155	156	8b	215	216
220	221	222	223	224	225	226	227	228	229	230	231	232	9a	233	234
160	161	162	163	164	165	166	167	168	169	170	171	172	9b	235	236
240	a0	241	a1	242	a2	243	a3	244	a4	245	a5	246	a6	247	a7
176	177	178	179	180	181	182	183	184	185	186	187	188	a8	248	a9
260	b0	261	b1	262	b2	263	b3	264	b4	265	b5	266	b6	267	b7
192	193	194	195	196	197	198	199	200	201	202	203	204	b8	271	b9
300	c0	301	c1	302	c2	303	c3	304	c4	305	c5	306	c6	307	c7
208	209	210	211	212	213	214	215	216	217	218	219	220	ca	313	cb
320	d0	321	d1	322	d2	323	d3	324	d4	325	d5	326	d6	327	d7
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
340	e0	341	e1	342	e2	343	e3	344	e4	345	e5	346	e6	347	e7
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
360	f0	361	f1	362	f2	363	f3	364	f4	365	f5	366	f6	367	f7
372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387

name: 8r-up1r8a at 11.0pt encoding: 8r mapping: 8r handling: default

Figure 8 The Palatino serif font in 8r (TeXBaseOne) encoding.

encoding, but also on the mapping from lower to uppercase characters, or more precise, on the existence of such a mapping.

Unless you want to play with these encodings and mappings, in most cases you can forget their details and rely on what other T_EX experts tell you to do. Normally switching from one to another encoding and/or mapping takes place with the change in fonts or when some special output encoding is needed, for instance in PDF annotations and/or unicode vectors that enable searching in documents. So, to summarize this: encodings and mappings depend on the fonts used as well have consequences for the language specific hyphenation patterns. Fortunately CON_TE_XT handles this for

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
000	001	002	003	004	005	006	007	008	009	00a	00b	00c	00d	00e	00f
016	017	018	019	020	021	022	023	024	025	026	027	028	029	030	031
032	033	034	035	036	037	038	039	040	041	042	043	044	045	046	047
048	049	050	051	052	053	054	055	056	057	058	059	060	061	062	063
064	065	066	067	068	069	070	071	072	073	074	075	076	077	078	079
080	081	082	083	084	085	086	087	088	089	090	091	092	093	094	095
096	097	098	099	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271
272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287
288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303
304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335
336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351
352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367
368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383
384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399

name: ec-up1r8a at 11.0pt encoding: ec mapping: ec handling: default

Figure 9 The Palatino serif font in ec (aka Cork) encoding.

you automatically.

If you want to know to what extend a font is complete and characters need to be composed on the fly, you can typeset a couple of tables. The (current) composition is shown by `\showaccents`:

— texnansi texnansi-lbr at 11.0pt: composed bottom char raw —

\ ' á b c d e f g h i j k l m n o p q r s t u v w x y z

Á B C D E F G H I J K L M N O P Q R S T U V W X Y Z

\ ' à b c d e f g h i j k l m n o p q r s t u v w x y z

```
name: texnansi-uplr8a at 11.0pt encoding: texnansi mapping: texnansi handling: default
```

Figure 10 The Palatino serif font in texnansi (YandY) encoding.

\^	À	Á	Â	Ã	Ä	Å	Æ	Ç	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ù	Ú	Û	Ü	Ý	Þ	ß
	à	á	â	ã	ä	å	æ	ç	ð	ñ	ò	ó	ô	õ	ö	×	ù	ú	û	ü	ý	þ	ß
	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	×	Ā	Ā	Ā	Ā	Ā	Ā	Ā
\~	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	×	ā	ā	ā	ā	ā	ā	ā
	Ă	Ă	Ă	Ă	Ă	Ă	Ă	Ă	Ă	Ă	Ă	Ă	Ă	Ă	Ă	×	Ă	Ă	Ă	Ă	Ă	Ă	Ă
	ă	ă	ă	ă	ă	ă	ă	ă	ă	ă	ă	ă	ă	ă	ă	×	ă	ă	ă	ă	ă	ă	ă
\"	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	Ä	×	Ä	Ä	Ä	Ä	Ä	Ä	Ä
	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	ä	×	ä	ä	ä	ä	ä	ä	ä
	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	×	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ
\H	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	×	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ	ȧ
	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	×	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ
	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	×	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ	Ȧ

\r	â	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	û	v	w	x	y	z
	Å	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	Û	V	W	X	Y	Z
\v	ă	ă	ć	ď	ě	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ
	Ă	Ĭ	Ć	Ď	Ě	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ
\u	ă	ă	ć	ď	ě	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ	ǝ
	Ă	Ĭ	Ć	Ď	Ě	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ	Ǟ
\=	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā
	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā
\.	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à	à
	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À
\b	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā
	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā
\d	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā
	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā
\k	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā
	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā
\c	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā	ā
	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā

with `\showcharacters`, you get an list of named characters (and glyphs) as known to the system.

	texnansi	texnansi-lbr	at 11.0pt:	composed	bottom	char	raw
`	textgrave	Ĉ	Ccircumflex	Ũ	Ucircumflex	Ỳ	Ygrave
´	textacute	ĉ	ccircumflex	û	ucircumflex	ỳ	ygrave
˘	textcaron	Ê	Ecircumflex	Ŵ	Wcircumflex	Ã	Atilde
˙	textbreve	ê	ecircumflex	ŵ	wcircumflex	ă	atilde
-	textmacron	Ĝ	Gcircumflex	Ŷ	Ycircumflex	Ĩ	Itilde
˚	textring	ĝ	gcircumflex	ŷ	ycircumflex	ĩ	itilde
ˆ	textcedilla	Ĥ	Hcircumflex	À	Agrave	Ñ	Ntilde
ˆ	textogonek	ĥ	hcircumflex	à	agrave	ñ	ntilde
ˆ	textbottomdot	Î	Icircumflex	È	Egrave	Õ	Otilde
ˆ	textcircumflex	î	icircumflex	è	egrave	õ	otilde
˙	textdotaccent	Ĵ	Jcircumflex	Ì	Igrave	Ŭ	Utilde
˙	texthungarumlaut	ĵ	jcircumflex	ì	igrave	ũ	utilde
˜	texttilde	Ô	Ocircumflex	Ò	Ograve	Ä	Adiaeresis
˜	textdiaeresis	ô	ocircumflex	ò	ograve	ä	adiaeresis
Â	Acircumflex	Ŝ	Scircumflex	Ù	Ugrave	Ë	Ediaeresis
â	acircumflex	ŝ	scircumflex	ù	ugrave	ë	ediaeresis

İ	Idiaeresis	ċ	cdotaccent	Ę	Eogonek	Ÿ	ycaron
ï	idiaeresis	Ė	Edotaccent	ę	eogonek	Ž	Zcaron
Ö	Odiaeresis	è	edotaccent	Į	Iogonek	ž	zcaron
ö	odiaeresis	Ġ	Gdotaccent	į	iogonek	ı	dotlessi
Ü	Udiaeresis	ğ	gdotaccent	U	Uogonek	ĵ	dotlessj
ü	udiaeresis	İ	Idotaccent	u	uogonek	ı	dotlessI
Ÿ	Ydiaeresis	ı	idotaccent	Ą	Aring	Ĵ	dotlessJ
ÿ	ydiaeresis	Ž	Zdotaccent	ą	aring	Æ	AEligature
Á	Aacute	ž	zdotaccent	Ů	Uring	æ	aeligature
á	aacute	Ā	Amacron	ů	uring	Ł	Lstroke
Ć	Cacute	ā	amacron	Ǻ	Abreve	ł	lstroke
ć	cacute	Ē	Emacron	ǻ	abreve	Ø	Ostroke
É	Eacute	ē	emacron	Ǽ	Ebreve	ø	ostroke
é	eacute	Ī	Imacron	ǿ	ebreve	Œ	OEligature
Í	Iacute	ī	imacron	Ǿ	Gbreve	œ	oeligature
í	iacute	Ō	Omacron	ǿ	gbreve	Š	Ssharp
Ĺ	Lacute	ō	omacron	ǐ	Ibreve	ß	ssharp
ĺ	lacute	Ū	Umacron	ǫ	ibreve	ıĴ	ıĴligature
Ň	Nacute	ū	umacron	Ǫ	Obreve	ıĵ	ıĵligature
ň	napute	Ç	Ccedilla	ǫ	obreve	ä	aumlaut
Ó	Oacute	ç	ccedilla	ǫ	ubreve	ë	eumlaut
ó	oacute	Ķ	Kcedilla	ǫ	ubreve	ï	iumlaut
Ř	Racute	ķ	kcedilla	Č	Ccaron	ö	oumlaut
ř	racute	ļ	lcedilla	č	ccaron	ü	uumlaut
Ś	Sacute	ļ	lcedilla	ǰ	Dcaron	Ä	Aumlaut
ś	sacute	Ņ	Ncedilla	ǰ	dcaron	Ě	Eumlaut
Ú	Uacute	ņ	ncedilla	Ě	Ecaron	Ī	Iumlaut
ú	uacute	Ŕ	Rcedilla	ě	ecaron	Ö	Oumlaut
Ý	Yacute	ŗ	rcedilla	Ľ	Lcaron	Ü	Uumlaut
ý	yacute	Ş	Scedilla	ĺ	lcaron	Ł	Lslash
Ž	Zacute	ş	scedilla	Ň	Ncaron	ł	lslash
ž	zacute	ţ	Tcedilla	ň	ncaron	Đ	Dslash
Đ	Dstroke	ţ	tcedilla	ř	rcaron	đ	dslash
đ	dstroke	Ő	Ohungarumlaut	ř	rcaron	Ø	Oslash
H	Hstroke	ő	ohungarumlaut	Š	Scaron	ø	oslash
h	hstroke	Ű	Uhungarumlaut	š	scaron	Š	Eszett
T	Tstroke	ű	uhungarumlaut	ť	Tcaron	ß	eszett
t	tstroke	Ą	Aogonek	ť	tcaron	Þ	Thorn
Ć	Cdotaccent	ą	aogonek	Ÿ	Ycaron	þ	thorn

If you want to know what patterns are used, you can try to hyphenate a word with `\showhyphenations`.

```
language : en(code:2)
font      : texnansi-lbr at 11.0pt
encoding  : texnansi
mapping   : texnansi
sample    : abra-cadabra
```

8 Regimes

When you key in an english document, a normal QWERTY keyboard combined with the standard ASCII character set will do. However, in many countries dedicated keyboards and corresponding input encodings are used. This means that certain keystrokes correspond to non standard ASCII characters and these need to be mapped onto the characters present in the font. Unless the input encoding matches the output (font) encoding, intermediate steps are needed to take care of the right mapping. For instance, input code 145 can become command `\eacute` which can result in character 123 of a certain font.

Although all kind of intermediate, direct or indirect, mappings are possible, in `CONTEXT` the preferred method is to go by named glyphs. The advantage of this method is that we can rather comfortably convert the input stream into different output streams as needed for typesetting text (the normal `TEX` process) and embedding information in the file (like annotations or font vectors needed for searching documents).

The conversion from input characters into named glyphs is handled by regimes. While further mapping is done automatically and is triggered by internal processes, regimes need to be chosen explicitly. This is because only the user knows what he has input.

Most encodings (like `i12`) have an associated regime. In addition there are a couple of platform dependent ones:

regime	platform
ibm	the old standard MSDOS code page
win	the western europe MS WINDOWS code page

9 Font handling

In the following fake paragraph, you can see a hyphenation point, a secondary sentence, separated by a comma, and a last sentence, ending with a period. Miraculously,

this paragraph fits into lines. Although exaggerated, these lines demonstrate that visually the hyphen and punctuation characters make the margin look ragged.



Before computers started to take over the traditional typesetter, it was common practice to move hyphens and punctuation into the margin, like in:



In this alternative, the margin looks less ragged, and this becomes more noticeable once you get aware of this phenomena.

Sometimes, shifting the characters completely into the margin is too much for the sensitive eye, for instance with a slanted font, where the characters already hang to the right. In such cases, we need to compromise.



PDF_T_EX has provisions to move characters into the margin when they end up at the end of a line. Such characters are called protruding characters. PDF_T_EX takes protruding into account when breaking a paragraph.³

³ One can also protrude characters without interference with the breaking routines, but since we consider this less optimal, CON_T_E_XT simply does not support it.

We demonstrate protruding using a quote from Hermann Zapf's article "About micro-typography and the *hz*-program" in *Electronic Publishing*, vol 6 (3), 1993.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

After T_EX has typeset this paragraph, it has constructed the following lines.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

As you can see, the height and depth of the lines depend on the characters, but their width equals what T_EX calls `\hspace`. However, the natural width of the lines may differ from `\hspace`.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

Here the inter-word space is fixed to what T_EX considers to be a space. This example also demonstrates that T_EX does not have spaces, but stretches the white area between words to suit its demands. When breaking lines, T_EX's mind is occupied by boxes, glue and penalties, or in more common language: (parts of) words, stretchable white space, and more or less preferred breakpoints.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the in-

struction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic

design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

This time we have enabled PDF_T_EX's protruding mechanism. The characters that stick into the margin are taken into account when breaking the paragraph into lines, but in the final result, they do not count in the width. Here we used an ugly three column layout so that we got a few more hyphens to illustrate the principle, but in the next examples we will stick to two columns.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not

so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

In this first example we just typeset the text in the traditional way. The hyphens and punctuation fit into the margin.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not

so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

In this example, the protruding machinery is put to labour. The hyphens and punctuation may now stick into the margin completely. The next two examples shows what happens when we limit the protruding to 75% and 50% respectively.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not

so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

Coming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer maga-

zines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between

good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely-praised program, called up on the screen, will make everything automatic from now on.

Although available in PDF_TE_X, this feature is not limited to PDF output. If this may comfort you: when protruding is not enabled the output is 100% identical.

Since protruding is related to the shape of the font, in CON_TE_XT setting it up (currently) takes place in a similar way as encodings.

```
\definefontsynonym
  [Lucida-Bright] [lbr] [encoding=texnansi,handling=pure]
```

We can also specify a handler at a higher level of abstraction:

```
\setupfontsynonym [Lucida-Bright] [handling=pure]
```

Or even:

```
\setupfontsynonym [Serif] [handling=pure]
```

The values that Hàn Thê Thành mentions in his thesis, are available under the names `punctuation` and `alpha`:

vector	protruding character set
<code>punctuation</code>	hyphenation and punctuation characters
<code>alpha</code>	the latin alphabet characters
<code>extended</code>	the other characters

The `extended` category maps the special characters onto the base 26 ones, and thereby honors the font encoding vector.

The default punctuation vector looks like this:

```
\startfonthandling[punctuation]
  \defineprotrudefactor , 0 .7
  \defineprotrudefactor . 0 .7
  \defineprotrudefactor : 0 .5
  \defineprotrudefactor ; 0 .5
  .....
\stopfonthandling
```

Equally valid are definitions like:

```
\defineprotrudefactor hyphen 0 .7
```

Any character can protrude:

```
\defineprotrudefactor A .05 .05
```

For convenience we let composed characters inherit the values from their parents.

```
\inheritprotrudefactor Acircumflex A
```

Instead of using numbers (like the 700 in the previous definition), we use fractions, one for the left and one for the right shift. There are a few more vectors defined, like `defalph` for characters, where we only apply very small shifts.

A combination of such vectors is packaged in a font handler using the following command:

```
\definefonthandling [normal] [punctuation,alpha]
```

These definitions depend on the encoding, they are loaded with the `\usehandling` command. You can for instance load the default definitions with: :

```
\usehandling[def]
```

Since this vector is already preloaded in `CONTEXT`, you normally don't have to provide this command.

In `CONTEXT` we have integrated protruding characters (hanging punctuation) it into the normal alignment macros.

```
\setupalign[hanging]
```

Font protruding is turned off automatically in controlled situations, and more control will be added in due time. We already mentioned that the amount of protruding depends on the shape, which is why we may need different values for slanted and bold shapes. One way of using the same vector for different shapes, is changing the strength of the protruding:

```
\definefonthandling [slanted] [punctuation] [right=1.5]
```

Here, because a slanted glyph already sticks into the margin, we limit protruding to punctuation. You may expect more (probably incompatible) fine tuning in the future. For the moment, when you want to play safe, the most simple way to enable hanging punctuation is to use the predefined `typescript`:

```
\usetypescript [serif,sans,mono] [hanging] [pure]
\setupalign[hanging]
```

or even:

```
\setupfontsynonym [handling=pure] \setupalign[hanging]
```

10 Math collection

Math is a complicated matter and therefore we will not spend that many words on the gory details. For the user it is enough to know that you can mix different math fonts in a comfortable way and that CONTEX T will take care of the proper mapping on specific math fonts.

Because the wide range of math symbols can come from different fonts, math characters are organized into so called math collections. Normally such a collection is chosen automatically when you load a font definition, just as with font encodings. The AMS math fonts extend the default math collection, which gives you a comfortable fall back. More information can be found in the documentation of the math module.

You can generate a list of the current math character set with the command `\showmathcharacters`.

— math characters —					
α	1 alpha	ρ	1 rho	Γ	3 Gamma
β	1 beta	σ	1 sigma	Δ	3 Delta
γ	1 gamma	τ	1 tau	E	3 Epsilon
δ	1 delta	υ	1 upsilon	Z	3 Zeta
ϵ	1 epsilon	ϕ	1 phi	H	3 Eta
ζ	1 zeta	χ	1 chi	Θ	3 Theta
η	1 eta	ψ	1 psi	I	3 Iota
θ	1 theta	ω	1 omega	K	3 Kappa
ι	1 iota	ε	1 varepsilon	Λ	3 Lambda
κ	1 kappa	ϑ	1 vartheta	M	3 Mu
λ	1 lambda	ϖ	1 varpi	N	3 Nu
μ	1 mu	ϱ	1 varrho	Ξ	3 Xi
ν	1 nu	ς	1 varsigma	O	3 Omicron
ξ	1 xi	φ	1 varphi	Π	3 Pi
o	1 omicron	A	3 Alpha	R	3 Rho
π	1 pi	B	3 Beta	Σ	3 Sigma

τ	3 Tau	\sum	3 sum	\propto	2 propto
Υ	3 Upsilon	\otimes	3 bigotimes	\sqsubseteq	2 sqsubseteq
Φ	3 Phi	\oplus	3 bigoplus	\sqsupseteq	2 sqsupseteq
χ	3 Chi	\odot	3 bigodot	\parallel	2 parallel
Ψ	3 Psi	\oint	3 ointop	$ $	2 mid
Ω	3 Omega	\sqcup	3 bigscup	\dashv	2 dashv
\aleph	2 aleph	\int	2 smallint	\vdash	2 vdash
ι	1 imath	\triangleleft	1 triangleleft	\nearrow	2 nearrow
j	1 jmath	\triangleright	1 triangleright	\searrow	2 searrow
ℓ	1 ell	\triangleup	2 bigtriangleup	\nwarrow	2 nwarrow
\wp	1 wp	∇	2 bigtriangledown	\swarrow	2 swarrow
\Re	2 Re	\wedge	2 wedge	\Leftrightarrow	2 Leftrightarrow
\Im	2 Im	\vee	2 vee	\Leftarrow	2 Leftarrow
∂	1 partial	\cap	2 cap	\Rightarrow	2 Rightarrow
∞	2 infty	\cup	2 cup	\leq	2 leq
$'$	2 prime	\ddagger	2 ddagger	\geq	2 geq
\emptyset	2 emptyset	\dagger	2 dagger	\succ	2 succ
∇	2 nabla	\sqcap	2 sqcap	\prec	2 prec
\top	2 top	\sqcup	2 sqcup	\approx	2 approx
\bot	2 bot	\oplus	2 uplus	\succeq	2 succeq
\triangle	2 triangle	\amalg	2 amalg	\preceq	2 preceq
\forall	2 forall	\diamond	2 diamond	\supset	2 supset
\exists	2 exists	\bullet	2 bullet	\subset	2 subset
\neg	2 neg	\wr	2 wr	\supseteq	2 supseteq
\sqcup	2 flat	\div	2 div	\sqsubseteq	2 subseteq
\natural	1 natural	\odot	2 odot	\in	2 in
\sharp	1 sharp	\oslash	2 oslash	\ni	2 ni
\clubsuit	2 clubsuit	\otimes	2 otimes	\gg	2 gg
\diamondsuit	2 diamondsuit	\ominus	2 ominus	\ll	2 ll
\heartsuit	2 heartsuit	\oplus	2 oplus	$/$	2 not
\spadesuit	2 spadesuit	\mp	2 mp	\leftrightarrow	2 leftrightarrow
\coprod	3 coprod	\pm	2 pm	\leftarrow	2 leftarrow
\bigvee	3 bigvee	\circ	2 circ	\rightarrow	2 rightarrow
\bigwedge	3 bigwedge	\bigcirc	2 bigcirc	\mapsto	2 mapstochar
\biguplus	3 biguplus	\setminus	2 setminus	\sim	2 sim
\bigcap	3 bigcap	\cdot	2 cdot	\simeq	2 simeq
\bigcup	3 bigcup	$*$	2 ast	\perp	2 perp
\int	3 intop	\times	2 times	\equiv	2 equiv
\prod	3 prod	\star	1 star	\asymp	2 asymp

˘	1 smile	→	1 vec	\	2 backslash
ˆ	1 frown	·	0 dot	}	2 rangle
⤵	1 leftharpoonup	~	3 widetilde	<	2 langle
⤴	1 leftharpoondown	^	3 widehat	}	2 rbrace
⤵	1 rightharpoonup	ˆ	3 lmoustache	{	2 lbrace
⤴	1 rightharpoondown	ˆ	3 rmoustache]	2 rceil
ˆ	1 lhook	(0 lgroup	[2 lceil
ˆ	1 rhook)	0 rgroup]	2 rfloor
.	1 ldotp		2 arrowvert]	2 lfloor
·	2 cdotp		2 Arrowvert	√	2 sqrt
:	0 colon	,	3 bracevert	†	2 dag
ˆ	0 acute		2 Vert	‡	2 ddag
`	0 grave		2 vert	§	2 S
..	0 ddot	↑	2 uparrow	¶	2 P
~	0 tilde	↓	2 downarrow	○	2 Orb
-	0 bar	↑	2 updownarrow	.	1 mathperiod
˘	0 breve	↑	2 Uparrow	.	1 textperiod
˘	0 check	↓	2 Downarrow	,	1 mathcomma
^	0 hat	↕	2 Updownarrow	,	1 textcomma

11 Predefined typefaces

We have predefined a couple of typeface combinations. If you want to define your own typefaces, you can peek into the following files:

script	file	content
type-enc		file name definitions, often encoding specific
type-syn		symbolic font names, often based on foundry names
type-map		PDF _T _E X map file definitions
type-spe		special definitions, like math collections
type-siz		size specifications
type-pre		predefined typescripts, downward compatible with <code>font-*.tex</code>
type-exa		example definitions, that can be used too

These files will be extended depending on user input and wishes, so feel free to submit additional definitions. The file with example definitions is probbaly the best place to start. There you will find for instance the `times` typeface.⁴

We will also add support for MathTimes with `texnansi` encodings.

```
\usetypescript[times][ec]
\switchtobodyfont[times,11pt]
```

This is a mixture of Times Roman (**serif**), Helvetica (**sans**, Computer Modern Roman (**mono**) and TX Times (**math**).

Due to relative scaling, these fonts combine in an acceptable way:

In ConT_EXt, fonts can be used in any combination, but in practice only certain combinations make sense. In most cases, you will use a Serif or Sans Serif font for the main body text. If you combine both shapes, you should be aware that not all combinations look well. A third shape is tagged as Mono. A mono spaced font is often used to identify text that is to be keyed in verbatim in computer programs. And then there is Math. Quite often $m + a + t + h \neq t + e + x + t$ although the normal text and numbers, as in $y = 2 \sin x$, in most cases is the same as the main body font.

12 Symbols and glyphs

Some day you may want to define your own symbols, if possible in such a way that they nicely adapt themselves to changes in style and size. A good example are the eurosymbols. You can take a look in `symb-eur.tex` to see how such a glyph is defined.

```
\definefontsynonym [EuroSerif]      [eurose]
\definefontsynonym [EuroSerifBold]  [euroseb]
...
\definefontsynonym [EuroSans]       [eurosa]
\definefontsynonym [EuroSansBold]   [eurosab]
...
\definefontsynonym [EuroMono]       [euromo]
\definefontsynonym [EuroMonoBold]   [euromob]
```

Here we use the free Adobe euro fonts, but there are alternatives available. The symbol itself is defined as:

```
\definesymbol [euro] [\getglyph{Euro}{\char160}]
```

You may notice that we only use the first part of the symbolic name. ConT_EXt will complete this name according to the current style. You can now access this symbol with `\symbol [euro]`

	<code>\tf</code>	<code>\bf</code>	<code>\sl</code>	<code>\it</code>	<code>\bs</code>	<code>\bi</code>
Serif	€	€	€	€	€	€

Sans	€	€	€	€	€	€
Mono	€	€	€	€	€	€

More details on defining symbols and symbol sets can be found in the reference manual and documentation of the symbol modules.

13 Map files

If you're already sick of reading about fonts, you probably don't want read this section. But alas, DVI post processors and PDF_T_EX will not work well if you don't provide them `map` files that tell them how to handle the files that contain the glyphs.

In its simplest form, a definition looks as follows:

```
usedname < realname.pfb texnansi.enc
```

This means as much as: when you want to include a file that has the `tfm` file `usedname`, take the outline file `realname.pfb` and embed it with the `texnansi` encoding vector. Sometimes you need more complicated directives and you can leave that to the experts.

14 Installing fonts

Most _T_EX distributions come with a couple of fonts, most noticeably the Computer Modern Roman typefaces. In order to use a font, _T_EX has to know its characteristics. These are defined in `tfm` and `vf` files. In addition to these files, on your system you can find a couple of more file types.

suffix	content
<code>tfm</code>	_T _E X specific font metric files that, in many cases, can be generated from <code>afm</code> files
<code>vf</code>	virtual font files, used for building glyph collections from other ones
<code>afm</code>	Adobe font metric files that are more limited than <code>tfm</code> files (especially for math fonts)
<code>pfm</code>	Windows specific font metric files, not used by _T _E X applications
<code>pfb</code>	files that contain the outline specification of the glyphs fonts, also called Type 1
<code>enc</code>	files with encoding vector specifications
<code>map</code>	files that specify how and what font files are to be included

On your disk (or cdrom) these files are organized in such a way that they can be located fast.⁵ The directory structure normally is as follows:

```
texmf / fonts / tfm      / vendor / name / *.tfm
           / afm      / vendor / name / *.afm
           / pfm      / vendor / name / *.pfm
           / vf       / vendor / name / *.vf
           / type1    / vendor / name / *.pfb
      / pdftex / config /                *.cfg
           / config /                *.map
           / config / encoding /        *.enc
```

The `texmf-local` or even better `texmf-fonts` tree normally contains your own fonts, so that you don't have to reinstall them when you reinstall the main tree. The `pdftex` directory contains the files that PDF_T_EX needs in order to make decisions about the fonts to include. The `enc` files are often part of distributions, as is the configuration `cfg` file. When you install new fonts, you often also have to add or edit `map` files.

CON_T_EX_T comes with a PERL script `texfont.pl` that you can use to install new fonts. Since its usage is covered by a separate manual, we limit ourselves to a short overview.

Say that you have just bought a new font. A close look at the files will reveal that you got at least a bunch of `afm` and `pfb` files and if you're lucky `tfm` files.

Installing such a font can be handled by this script. For this you need to know (or invent) the name of the font vendor, as well as the name of the font. The full set of command line switches is given below:⁶

switch	meaning
fontroot	texmf font root (automatically determined)
vendor	vendor name (first level directory)
collection	font collection (second level directory)
encoding	encoding vector (default: texnansi)
sourcepath	when installing, copy from this path
install	copy files from source to font tree
makepath	when needed, create the paths
show	run tex on *. <code>tex</code> afterwards

If you have installed T_E_X or F_P_T_EX (possibly from the T_EXlive CDROM) you will have many thousands of

⁵ font files on your system.

there are a couple of more switches described in the manual `mtexfonts`.

⁶

You seldom need to use them all. In any case it helps if you have a local path defined already. The next sequence does the trick:

```
texfont --ve=FontFun --co=FirstFont --en=texnansi --ma --in
```

This will generate the `tfm` files from the `afm` files, and copy them to the right place. The Type 1 files (`pfb`) will be copied too. The script also generates a `map` file. When this is done successfully, a \TeX file is generated and processed that shows the font maps. If this file looks right, you can start using the fonts. The \TeX file also show you how to define the fonts.

This script can also do a couple of more advanced tricks. Let us assume that we have bought (or downloaded) a new font packages in the files `demofont.afm` and `demofont.pfb` which are available on the current (probably scratch) directory. First we make sure that this font is installed (in our case we use \TeX 's informal font):

```
texfont --ve=test --co=test --ma --in demofont
```

We can now say:

```
\loadmapfile[texnansi-test-test.map]
\definefontsynonym[DemoFont][texnansi-demofont]
\ruledhbox{\definedfont[DemoFont at 50pt]Interesting}
```



From this font, we can derive a slanted alternative by saying:

```
texfont --ve=test --co=test --ma --in --sla=.167 demofont
```

The map file is automatically extended with the entry needed.

```
\definefontsynonym[DemoFont-Slanted][texnansi-demofont-slanted-167]
\ruledhbox{\definedfont[DemoFont-Slanted at 50pt]Interesting}
```



We can also create a wider version:

```
texfont --ve=test --co=test --ma --in --ext=1.50 demofont
```

When you use the `--make` and `--install` switch, the directories are made, fonts installed, and entries appended to the map file if needed.

```
\definefontsynonym[DemoFont-Extended][texnansi-demofont-extended-1500]
\ruledhbox{\definedfont[DemoFont-Extended at 50pt]Interesting}
```




Instead of using pseudo caps in \TeX by using `\kap`, you can also create a pseudo small caps font.

```
texfont --ve=test --co=test --ma --in --cap=0.75 demofont
```

This method is much more robust but at the cost of an extra font.

```
\definefontsynonym[DemoFont-Caps][texnansi-demofont-capitalized-750]
\ruledhbox{\definedfont[DemoFont-Caps at 50pt]Interesting}
```



switch	meaning
<code>extend=factor</code>	stretch the font to the given factor
<code>narrow=factor</code>	shrink the font to the given factor
<code>slant=factor</code>	create a slanted font
<code>caps=factor</code>	replace lowercase characters by small uppercase ones
<code>test</code>	use test/test as vendor/collection

When manipulating a font this way, you need to provide a file name. Instead of a factor you can give the keyword `default` or a `*`.

```
texfont --test --auto --caps=default demofont
```

The previous example runs create fonts with the rather verbose names:

```
demofont
demofont-slanted-167
demofont-extended-150
demofont-capitalized-750
```

This naming scheme makes it possible to use more instances without the risk of conflicts.

In the distribution you will find an example batch file `type-tmf.dat` which creates metrics for some free fonts for the encoding specified. When you create the default font metrics this way, preferably `texmf-fonts`, you have a minimal font system tuned for your preferred encoding without the risk for name clashes. When you also supply `--install`, the font outlines will be copied from the main tree to the fonts tree, which sometimes is handy from the perspective of consistency.

15 Getting started

This is the most tricky section to write, since fonts is probably the most complicated aspect of $\text{T}_{\text{E}}\text{X}$. If you are not sure that you have the latex $\text{CON}_{\text{T}}\text{E}_{\text{X}}\text{T}$, fetch the zipped archive `cont-tmf.zip` from the main site or one of the mirrors. Unzip the archive in the path `texmf-local` (or similar):

```
unzip cont-tmf.zip
```

Run `mktexlsr` afterwards. In order to prevent unwanted interference, you may want to remove old copies of $\text{CON}_{\text{T}}\text{E}_{\text{X}}\text{T}$. Removing the `base` paths is enough.

Next, take a look at `cont-sys.rme` for options that you can set. You need to copy this file into one with the suffix `tex` and put it in the $\text{CON}_{\text{T}}\text{E}_{\text{X}}\text{T}$ user path. If you already have such a file, make sure that you bring it up to date. Choose a default encoding and preload the map files that $\text{PDF}_{\text{T}}\text{E}_{\text{X}}$ needs. If you want to use the default filenames, load the `berry` typescript too, but if you are going to install your own fonts, you may use the more verbose naming scheme (which is what we do at PRAGMA ADE).

Again, make sure that you run `mktexlsr` after each file that you add to the system.

We already discussed `texfont` and if you have a couple of megabytes left, you can now generate font metrics. If you want to isolate your fonts from the main trees, which is a good idea if you also want to install your own fonts.

The way $\text{T}_{\text{E}}\text{X}$ searches for files (we're talking WEB_{2}C now) is determined by the configuration file to which the `TEXMFCNF` environment variable points (the following examples are from my own system):

```
set TEXMFCNF=T:/TEXMF/WEB2C
```

When searching for files, a list of directories is used:

```
set TEXMF={$TEXMFFONTS,$TEXMFLOCAL,!!$TEXMFMAIN}
```

Here we've added a font pat, which itself is set with:

```
set TEXMFMAIN=E:/TEX/TEXMF
set TEXMFLOCAL=E:/TEX/TEXMF-LOCAL
set TEXMFFONTS=E:/TEX/TEXMF-FONTS
```

Now you can generate metrics and map files. The batch file is searched for at the `CONTEXT` data path in the `TEXMF` tree or on the local path.

```
texfont --encoding=ec --batch type-tmf.dat
```

If you want to play with encoding, you can also generate more encodings, like `8r` or `texnansi`.

```
texfont --encoding=texnansi --batch type-tmf.dat
texfont --encoding=8r --batch type-tmf.dat
```

After a while, there will be generated `tfm`, `vf`, and `map` files. If you let `CONTEXT` pass the map file directives to `PDFTEX`, you're ready now. Otherwise you need to add the names of the mapfiles to the file `pdfTEX.cfg`. You can best add them in front of the list, and, if you use `CONTEXT` exclusively, you can best remove the other ones.

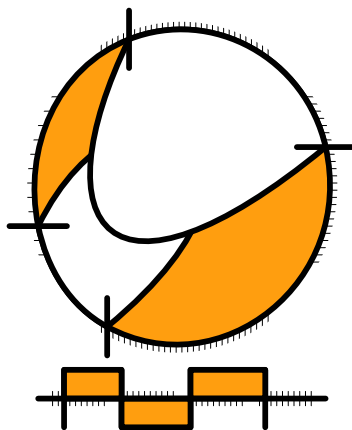
As a test you can process the `TEX` files that are generated in the process. These also give you an idea of how well the encoding vectors match your expectations.

Now, the worst that can happen to you when you process your files, is that you get messages concerning unknown `tfm` files or reports on missing fonts when `PDFTEX` writes the file. In that case, make sure that you indeed *have* the right fonts (generated) and/or that the map files are loaded. As a last resort you can load all map files by saying:

```
\usetypscript [map] [all]
```

and take a look at the log file and see what is reported.

In due time we will provide font generation scripts for installation of other fonts as well as extend the typescript collection.



PRAGMA

Advanced Document Engineering | Ridderstraat 27 | 8061GH Hasselt NL
tel: +31 (0)38 477 53 69 | email: pragma@wxs.nl | internet: www.pragma-ade.com