Fonts in ConTeXt

Hans Hagen

April 2001

Contents

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.

2 Fonts files and encodings

In ConTeXt 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 [fig:environment].

```
\showbodyfontenvironment % [lbr]
```

[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:

```
\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]
```

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

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 ConTeXt 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 ans 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?

[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 is *fond* of fonts?
Who claims that $t+e+x+t=m+a+t+h$?
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:

```
\usetypescript[modern][ec]   % for western european users
\usetypescript[modern][il2]  % for czech and slovak users
\usetypescript[modern][pl0]  % for polish users
```

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

```
\usetypescript[postscript][texnansi] % our prefered encoding
\usetypescript[postscript][ec]       % another popular one
```

It may not be clear form 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:

$\sqrt{625} = 5\alpha$
$\sqrt{\mathbf{625}} = \mathbf{5}\alpha$
$\sqrt{625} = \mathbf{5}\alpha$
$\sqrt{\mathbf{625}} = \mathbf{5}\alpha$

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:

$\sqrt{625} = 5\alpha$
$\sqrt{\mathbf{625}} = \mathbf{5}\alpha$

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

$\sqrt{625} = 5\alpha$
$\boldsymbol{\sqrt{625} = 5\alpha}$

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:

$\sqrt{625} = 5\alpha$
$\sqrt{\mathbf{625}} = \mathbf{5}\alpha$
$\sqrt{625} = \mathbf{5}\alpha$
$\sqrt{625} = 5\alpha$

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.

$\sqrt{625} = 5\alpha$
$\sqrt{\mathbf{625}} = \mathbf{5}\alpha$
$\sqrt{625} = \mathbf{5}\alpha$

$\sqrt{625} = 5\alpha$

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, TEX 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 TEX
texnansi	a combination of TEX and Adobe standard encoding
ec	a rather complete encoding suitable for most languages
il2	iso latin 2 encoding as needed for Czech and Slovak
pl0	a native Polish encoding

These encodings are font relate as is shown the the following tables:

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 TEX hyphenate words in the appropriate way. The hyphenation patterns are applied to the internal data

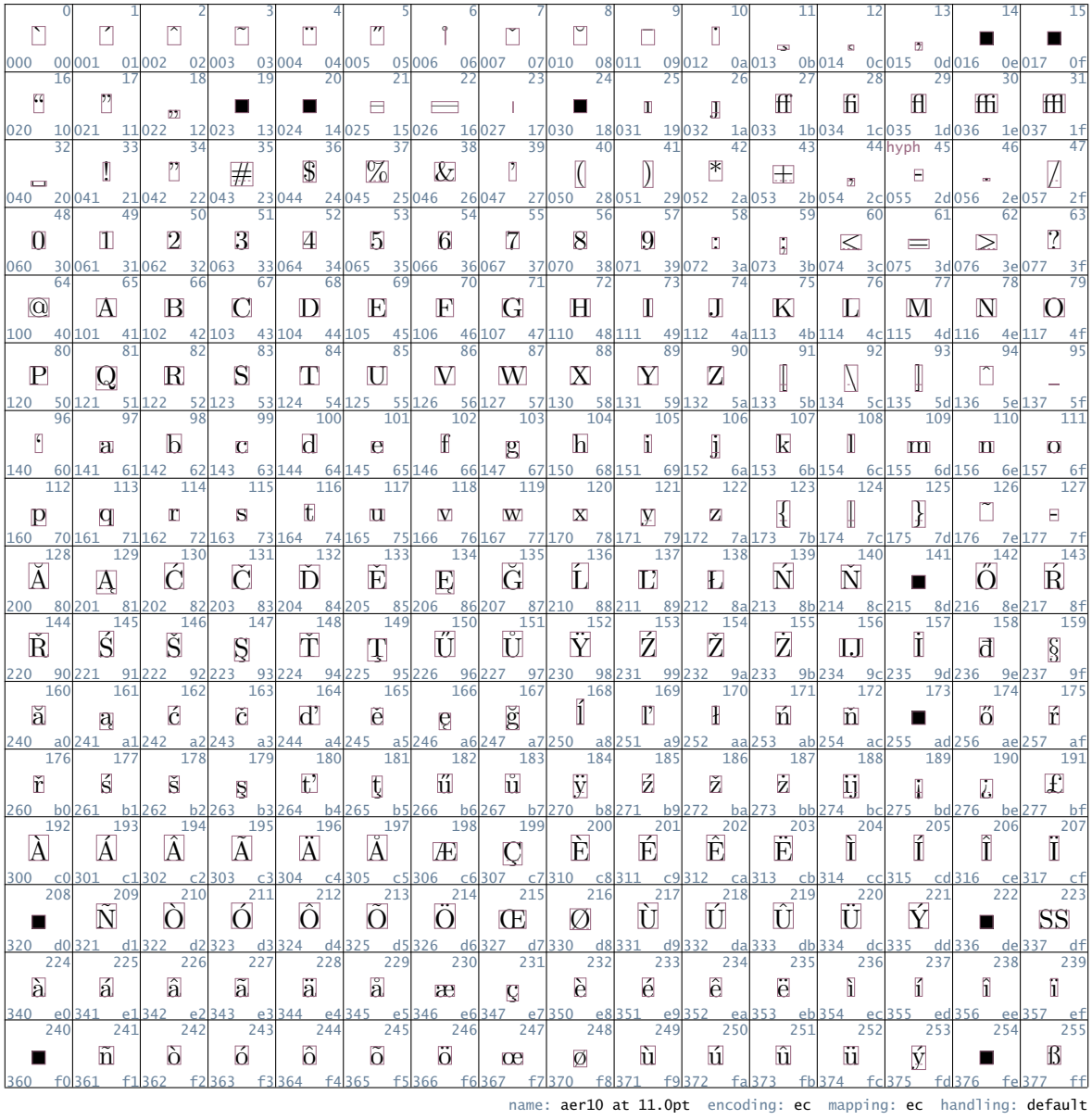


Figure 8 The output of `\showfont[aer10]`.

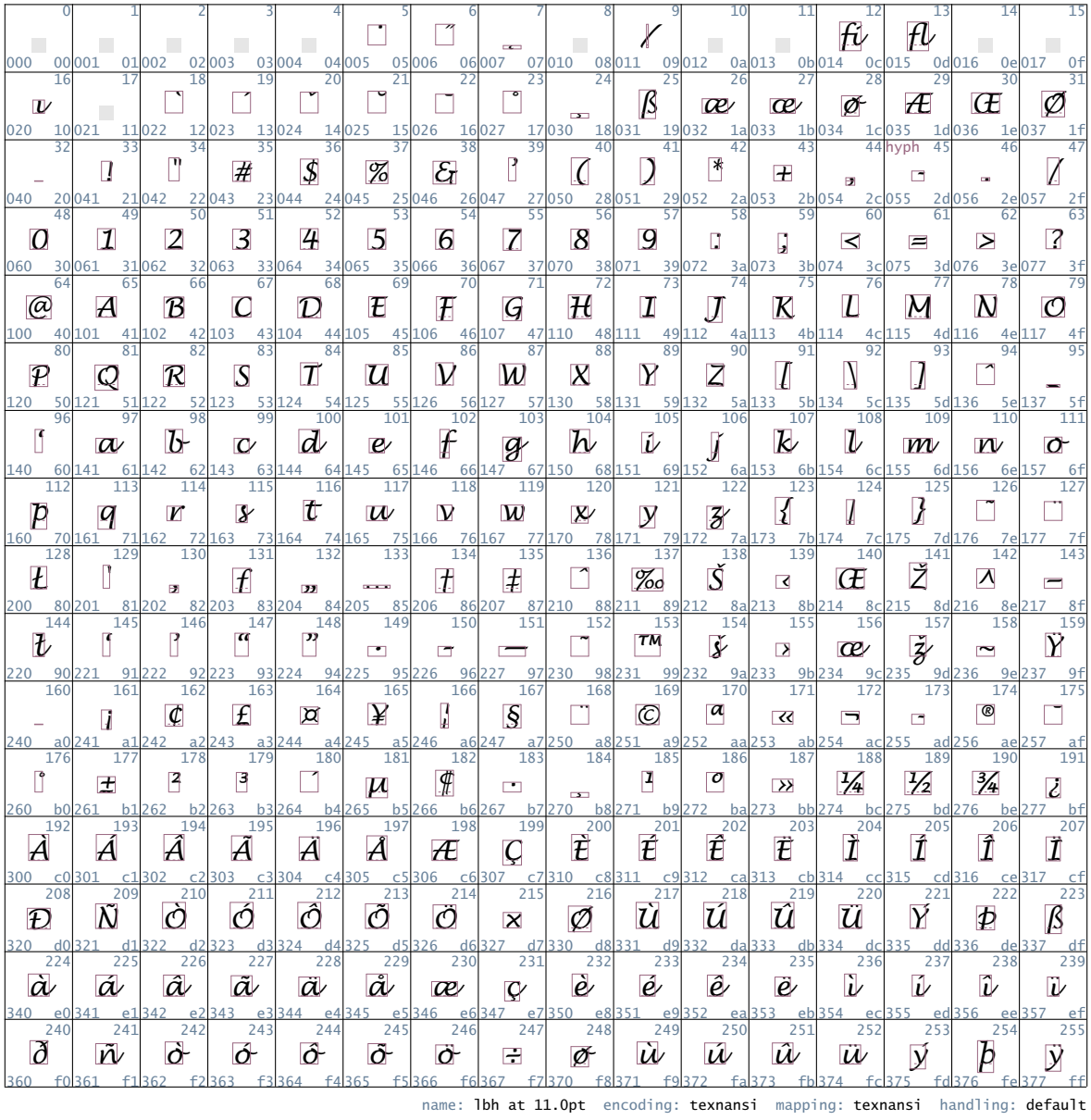


Figure 9 The output of `\showfont[lbh]`.

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 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 TEX 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 annota-

tions 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 ConTeXt handles this for 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 a couple of tables. The (current) composition is shown by `\showaccents`:

texnansi lbr at 11.0pt: composed bottom char raw

\'	á	b́	ć	d́	é	f́	ǵ	h́	í	j́	ḱ	ĺ	ḿ	ń	ó	ṕ	q́	ŕ	ś	t́	ú	v́	ẃ	x́	ý	ź
	Á	B́	Ć	D́	É	F́	Ǵ	H́	Í	J́	Ḱ	Ĺ	Ḿ	Ń	Ó	Ṕ	Q́	Ŕ	Ś	T́	Ú	V́	Ẃ	X́	Ý	Ź
\`	à	b̀	c̀	d̀	è	f̀	ǵ̀	h̀	ì	j̀	k̀	l̀	m̀	ǹ	ò	p̀	q̀	r̀	s̀	t̀	ù	v̀	ẁ	x̀	ỳ	ż
	À	B̀	C̀	D̀	È	F̀	Ǵ̀	H̀	Ì	J̀	K̀	L̀	M̀	Ǹ	Ò	P̀	Q̀	R̀	S̀	T̀	Ù	V̀	Ẁ	X̀	Ỳ	Ż
\^	â	b̂	ĉ	d̂	ê	f̂	ĝ	ĥ	î	ĵ	k̂	l̂	m̂	n̂	ô	p̂	q̂	r̂	ŝ	t̂	û	v̂	ŵ	x̂	ŷ	ẑ
	Â	B̂	Ĉ	D̂	Ê	F̂	Ĝ	Ĥ	Î	Ĵ	K̂	L̂	M̂	N̂	Ô	P̂	Q̂	R̂	Ŝ	T̂	Û	V̂	Ŵ	X̂	Ŷ	Ẑ
\~	ã	b̃	c̃	d̃	ẽ	f̃	g̃	h̃	ĩ	j̃	k̃	l̃	m̃	ñ	õ	p̃	q̃	r̃	s̃	t̃	ũ	ṽ	w̃	x̃	ỹ	z̃
	Ã	B̃	C̃	D̃	Ẽ	F̃	G̃	H̃	Ĩ	J̃	K̃	L̃	M̃	Ñ	Õ	P̃	Q̃	R̃	S̃	T̃	Ũ	Ṽ	W̃	X̃	Ỹ	Z̃
\"	ä	b̈	c̈	d̈	ë	f̈	g̈	ḧ	ï	j̈	k̈	l̈	m̈	n̈	ö	p̈	q̈	r̈	s̈	ẗ	ü	v̈	ẅ	ẍ	ÿ	z̈
	Ä	B̈	C̈	D̈	Ë	F̈	G̈	Ḧ	Ï	J̈	K̈	L̈	M̈	N̈	Ö	P̈	Q̈	R̈	S̈	T̈	Ü	V̈	Ẅ	Ẍ	Ÿ	Z̈
\H	á	b̋	ć	d̋	é	f̋	g̋	h̋	í	j̋	k̋	l̋	m̋	n̋	ő	p̋	q̋	r̋	s̋	t̋	ű	v̋	ẅ	x̋	ý	z̋
	Á	B̋	Ć	D̋	É	F̋	G̋	H̋	Í	J̋	K̋	L̋	M̋	N̋	Ő	P̋	Q̋	R̋	S̋	T̋	Ű	V̋	Ẅ	X̋	Ý	Z̋
\r	å	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	ů	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	Ů	V	W	X	Y	Z
\v	ǎ	b̌	č	ď	ě	f̌	ǧ	ȟ	ǐ	ǰ	ǩ	ľ	m̌	ň	ǒ	p̌	q̌	ř	š	ť	ǔ	v̌	w̌	x̌	y̌	ž
	Ǎ	B̌	Č	Ď	Ě	F̌	Ǧ	Ȟ	Ǐ	J̌	Ǩ	Ľ	M̌	Ň	Ǒ	P̌	Q̌	Ř	Š	Ť	Ǔ	V̌	W̌	X̌	Y̌	Ž
\u	ă	b̆	č	ď	ĕ	f̆	ğ	ȟ	ĭ	ǰ	ǩ	ľ	m̆	ň	ŏ	p̆	q̆	ř	š	ť	ŭ	v̆	w̆	x̆	y̆	ž
	Ă	B̆	Č	Ď	Ĕ	F̆	Ğ	Ȟ	Ĭ	J̆	Ǩ	Ľ	M̆	Ň	Ŏ	P̆	Q̆	Ř	Š	Ť	Ŭ	V̆	W̆	X̆	Y̆	Ž
\=	ā	b̄	c̄	d̄	ē	f̄	ḡ	h̄	ī	j̄	k̄	l̄	m̄	n̄	ō	p̄	q̄	r̄	s̄	t̄	ū	v̄	w̄	x̄	ȳ	z̄
	Ā	B̄	C̄	D̄	Ē	F̄	Ḡ	H̄	Ī	J̄	K̄	L̄	M̄	N̄	Ō	P̄	Q̄	R̄	S̄	T̄	Ū	V̄	W̄	X̄	Ȳ	Z̄
\.	ȧ	ḃ	ċ	ḋ	ė	ḟ	ġ	ḣ	i	j	k	l	ṁ	ṅ	ȯ	ṗ	q	ṙ	ṡ	ṫ	u	v	ẇ	ẋ	ẏ	ż
	Ȧ	Ḃ	Ċ	Ḋ	Ė	Ḟ	Ġ	Ḣ	İ	J	K	L	Ṁ	Ṅ	Ȯ	Ṗ	Q	Ṙ	Ṡ	Ṫ	U	V	Ẇ	Ẋ	Ẏ	Ż
\b	a̲	b̲	c̲	d̲	e̲	f̲	g̲	h̲	i̲	j̲	k̲	l̲	m̲	n̲	o̲	p̲	q̲	r̲	s̲	t̲	u̲	v̲	w̲	x̲	y̲	z̲
	A̲	B̲	C̲	D̲	E̲	F̲	G̲	H̲	I̲	J̲	K̲	L̲	M̲	N̲	O̲	P̲	Q̲	R̲	S̲	T̲	U̲	V̲	W̲	X̲	Y̲	Z̲
\d	ạ	ḅ	ç	ḍ	ẹ	f̣	g̣	ḥ	ị	đ	ḳ	ḷ	ṃ	ṇ	ọ	ṗ	q̣	ṛ	ṣ	ṭ	ụ	ṿ	ẉ	ẋ	ỵ	ẓ
	Ạ	Ḅ	Ç	Ḍ	Ẹ	F̣	G̣	Ḥ	Ị	J̣	Ḳ	Ḷ	Ṃ	Ṇ	Ọ	Ṗ	Q̣	Ṛ	Ṣ	Ṭ	Ụ	Ṿ	Ẉ	Ẋ	Ỵ	Ẓ
\k	ą	ḇ	ç	ḏ	ę	f̨	g̨	ḫ	į	j̨	ḳ	ḷ	ṃ	ṇ	ǫ	ṗ	q̨	ṛ	ṣ	ṭ	ų	ṿ	ẉ	ẋ	ỵ	ẓ
	Ą	Ḇ	Ç	Ḏ	Ę	F̨	G̨	Ḫ	Į	J̨	Ḳ	Ḷ	Ṃ	Ṇ	Ǫ	Ṗ	Q̨	Ṛ	Ṣ	Ṭ	Ų	Ṿ	Ẉ	Ẋ	Ỵ	Ẓ
\c	ą	ḇ	ç	ḏ	ę	f̧	g̨	ḫ	į	j̨	ḳ	ḷ	ṃ	ṇ	ǫ	ṗ	q̨	ṛ	ṣ	ṭ	ų	ṿ	ẉ	ẋ	ỵ	ẓ

Ạ Ḅ Ç Ḍ Ẹ Ḟ Ḡ Ḥ Ị Ĵ Ḳ Ḷ Ṃ Ṇ Ọ Ṗ Ǫ Ṛ Ṣ Ṭ Ụ Ṿ Ẉ Ẋ Ỵ Ẓ

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

texnansi lbr at 11.0pt: composed bottom char raw

`	textgrave	Û	Ucircumflex	Ï	Idiaeresis	D	Dstroke
´	textacute	û	ucircumflex	ï	idiaeresis	d	dstroke
ˇ	textcaron	Ŵ	Wcircumflex	Ö	Odiaeresis	H	Hstroke
˘	textbreve	ŵ	wcircumflex	ö	odiaeresis	h	hstroke
¯	textmacron	Ŷ	Ycircumflex	Ü	Udiaeresis	T	Tstroke
°	textring	ŷ	ycircumflex	ü	udiaeresis	t	tstroke
¸	textcedilla	À	Agrave	Ÿ	Ydiaeresis	Ċ	Cdotaccent
˛	textogonek	à	agrave	ÿ	ydiaeresis	ċ	cdotaccent
.	textbottomdot	È	Egrave	Á	Aacute	Ė	Edotaccent
ˆ	textcircumflex	è	egrave	á	aacute	ė	edotaccent
˙	textdotaccent	Ì	Igrave	Ć	Cacute	Ġ	Gdotaccent
˝	texthungarumlaut	ì	igrave	ć	cacute	ġ	gdotaccent
~	texttilde	Ò	Ograve	É	Eacute	İ	Idotaccent
¨	textdiaeresis	ò	ograve	é	eacute	i	idotaccent
Â	Acircumflex	Ù	Ugrave	Í	Iacute	Ż	Zdotaccent
â	acircumflex	ù	ugrave	í	iacute	ż	zdotaccent
Ĉ	Ccircumflex	Ỳ	Ygrave	Ĺ	Lacute	Ā	Amacron
ĉ	ccircumflex	ỳ	ygrave	ĺ	lacute	ā	amacron
Ê	Ecircumflex	Ã	Atilde	Ń	Nacute	Ē	Emacron
ê	ecircumflex	ã	atilde	ń	nacute	ē	emacron
Ĝ	Gcircumflex	Ĩ	Itilde	Ó	Oacute	Ī	Imacron
ĝ	gcircumflex	ĩ	itilde	ó	oacute	ī	imacron
Ĥ	Hcircumflex	Ñ	Ntilde	Ŕ	Racute	Ō	Omacron
ĥ	hcircumflex	ñ	ntilde	ŕ	racute	ō	omacron
Î	Icircumflex	Õ	Otilde	Ś	Sacute	Ū	Umacron
î	icircumflex	õ	otilde	ś	sacute	ū	umacron
Ĵ	Jcircumflex	Ũ	Utilde	Ú	Uacute	Ç	Ccedilla
ĵ	jcircumflex	ũ	utilde	ú	uacute	ç	ccedilla
Ô	Ocircumflex	Ä	Adiaeresis	Ý	Yacute	Ķ	Kcedilla
ô	ocircumflex	ä	adiaeresis	ý	yacute	ķ	kcedilla
Ŝ	Scircumflex	Ë	Ediaeresis	Ź	Zacute	Ļ	Lcedilla
ŝ	scircumflex	ë	ediaeresis	ź	zacute	ļ	lcedilla

Ņ	Ncedilla	ů	uring	Ř	Rcaron	ß	ssharp
ņ	ncedilla	Ă	Abreve	ř	rcaron	IJ	IJligature
Ŗ	Rcedilla	ă	abreve	Š	Scaron	ij	ijligature
ŗ	rcedilla	Ĕ	Ebreve	š	scaron	ä	aumlaut
Ş	Scedilla	ĕ	ebreve	Ť	Tcaron	ë	eumlaut
ş	scedilla	Ğ	Gbreve	ť	tcaron	ï	iumlaut
Ţ	Tcedilla	ğ	gbreve	Ў	Ycaron	ö	oumlaut
ţ	tcedilla	Ĭ	Ibreve	ў	ycaron	ü	uumlaut
Ő	Ohungarumlaut	ĭ	ibreve	Ž	Zcaron	Ä	Aumlaut
ő	ohungarumlaut	Ŏ	Obreve	ž	zcaron	Ë	Eumlaut
Ű	Uhungarumlaut	ŏ	obreve	i	dotlessi	Ï	Iumlaut
ű	uhungarumlaut	Ŭ	Ubreve	ȷ	dotlessj	Ö	Oumlaut
Ą	Aogonek	ŭ	ubreve	I	dotlessI	Ü	Uumlaut
ą	aogonek	Č	Ccaron	J	dotlessJ	Ł	Lslash
Ę	Eogonek	č	ccaron	Æ	AEligature	ł	lslash
ę	eogonek	Ď	Dcaron	æ	aeligature	D	Dslash
Į	Iogonek	ď	dcaron	Ł	Lstroke	d	dslash
į	iogonek	Ě	Ecaron	ł	lstroke	Ø	Oslash
Ų	Uogonek	ě	ecaron	Ø	Ostroke	ø	oslash
ų	uogonek	Ľ	Lcaron	ø	ostroke	SS	Eszett
Å	Aring	ľ	lcaron	Œ	OEligature	ß	eszett
å	aring	Ň	Ncaron	œ	oeligature	Þ	Thorn
Ů	Uring	ň	ncaron	SS	Ssharp	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 : 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 `il2`) 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 noticable 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.

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

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

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

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 TEX 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 TEX calls `\hsize`. However, the natural width of the lines may differ from `\hsize`.

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 TEX considers to be a space. This example also demonstrates that TEX does not have spaces, but stretches the white area between words to suit its demands. When breaking lines, TEX'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 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.

This time we have enabled pdfTEX'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 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.

Although available in pdfTeX, 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 ConTeXt 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 Thế Thành mentions in his thesis, are available under the names `punctuation` and `alpha`:

vector	protruding character set
`punctuation`	hyphenation and punctuation characters
`alpha`	the latin alphabet characters
`extended`	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
```

Equaly 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 [hanging=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 ConTeXt 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	A	3 Alpha	$\Im$	2 Im
β	1 beta	B	3 Beta	∂	1 partial
γ	1 gamma	Γ	3 Gamma	∞	2 infty
δ	1 delta	Δ	3 Delta	$\prime$	2 prime
ϵ	1 epsilon	E	3 Epsilon	$\emptyset$	2 emptyset
ζ	1 zeta	Z	3 Zeta	∇	2 nabla
η	1 eta	H	3 Eta	$\top$	2 top
θ	1 theta	Θ	3 Theta	$\bot$	2 bot
ι	1 iota	I	3 Iota	$\triangle$	2 triangle
κ	1 kappa	K	3 Kappa	$\forall$	2 forall
λ	1 lambda	Λ	3 Lambda	$\exists$	2 exists
μ	1 mu	M	3 Mu	$\neg$	2 neg
ν	1 nu	N	3 Nu	$\cup$	2 flat
ξ	1 xi	Ξ	3 Xi	$\natural$	1 natural
π	1 pi	Π	3 Pi	$\sharp$	1 sharp
ρ	1 rho	R	3 Rho	$\clubsuit$	2 clubsuit
σ	1 sigma	Σ	3 Sigma	$\diamondsuit$	2 diamondsuit
τ	1 tau	T	3 Tau	$\heartsuit$	2 heartsuit
υ	1 upsilon	Υ	3 Upsilon	$\spadesuit$	2 spadesuit
ϕ	1 phi	Φ	3 Phi	$\coprod$	3 coprod
χ	1 chi	X	3 Chi	$\bigvee$	3 bigvee
ψ	1 psi	Ψ	3 Psi	$\bigwedge$	3 bigwedge
ω	1 omega	Ω	3 Omega	$\biguplus$	3 biguplus
ε	1 varepsilon	$\aleph$	2 aleph	$\bigcap$	3 bigcap
ϑ	1 vartheta	$\imath$	1 imath	$\bigcup$	3 bigcup
ϖ	1 varpi	$\jmath$	1 jmath	$\int$	3 intop
ϱ	1 varrho	ℓ	1 ell	$\prod$	3 prod
ς	1 varsigma	$\wp$	1 wp	$\sum$	3 sum
φ	1 varphi	$\Re$	2 Re	$\bigotimes$	3 bigotimes

$\bigoplus$	3	bigoplus	$\sqsupseteq$	2	sqsupseteq	$\leftharpoonup$	1 leftharpoonup
$\bigodot$	3	bigodot	$\parallel$	2	parallel	$\leftharpoondown$	1 leftharpoondown
$\oint$	3	ointop	$\mid$	2	mid	$\rightharpoonup$	1 rightharpoonup
$\bigsqcup$	3	bigsqcup	$\dashv$	2	dashv	$\rightharpoondown$	1 rightharpoondown
$\int$	2	smallint	$\vdash$	2	vdash	‘	1 lhook
$\triangleleft$	1	triangleleft	$\nearrow$	2	nearrow	’	1 rhook
$\triangleright$	1	triangleright	$\searrow$	2	searrow	.	1 ldotp
$\bigtriangleup$	2	bigtriangleup	$\nwarrow$	2	nwarrow	$\cdot$	2 cdotp
$\bigtriangledown$	2	bigtriangledown	$\swarrow$	2	swarrow	:	0 colon
$\wedge$	2	wedge	$\nleftrightarrow$	2	Leftrightarrow	$\acute{}$	0 acute
$\vee$	2	vee	$\Leftarrow$	2	Leftarrow	$\grave{}$	0 grave
$\cap$	2	cap	$\Rightarrow$	2	Rightarrow	$\ddot{}$	0 ddot
$\cup$	2	cup	$\leq$	2	leq	$\tilde{}$	0 tilde
$\ddagger$	2	ddagger	$\geq$	2	geq	$\bar{}$	0 bar
$\dagger$	2	dagger	$\succ$	2	succ	$\breve{}$	0 breve
$\sqcap$	2	sqcap	$\prec$	2	prec	$\check{}$	0 check
$\sqcup$	2	sqcup	$\approx$	2	approx	$\hat{}$	0 hat
$\uplus$	2	uplus	$\succeq$	2	succeq	$\vec{}$	1 vec
$\amalg$	2	amalg	$\preceq$	2	preceq	$\dot{}$	0 dot
$\diamond$	2	diamond	$\supset$	2	supset	$\widetilde{}$	3 widetilde
$\bullet$	2	bullet	$\subset$	2	subset	$\widehat{}$	3 widehat
$\wr$	2	wr	$\supseteq$	2	supseteq	$\overgroup{}$	3 lmoustache
$\div$	2	div	$\subseteq$	2	subseteq	$\undergroup{}$	3 rmoustache
$\odot$	2	odot	$\in$	2	in	(	0 lgroup
$\oslash$	2	oslash	$\ni$	2	ni	)	0 rgroup
$\otimes$	2	otimes	$\gg$	2	gg	$\vert$	2 arrowvert
$\ominus$	2	ominus	$\ll$	2	ll	$\Vert$	2 Arrowvert
$\oplus$	2	oplus	/	2	not	$\arrowvert$	3 bracevert
$\mp$	2	mp	$\leftrightarrow$	2	leftrightarrow	$\Vert$	2 Vert
$\pm$	2	pm	$\leftarrow$	2	leftarrow	$\vert$	2 vert
$\circ$	2	circ	$\rightarrow$	2	rightarrow	$\uparrow$	2 uparrow
$\bigcirc$	2	bigcirc	$\mapsto$	2	mapstochar	$\downarrow$	2 downarrow
\	2	setminus	$\sim$	2	sim	$\updownarrow$	2 updownarrow
$\cdot$	2	cdot	$\simeq$	2	simeq	$\Uparrow$	2 Uparrow
$\ast$	2	ast	$\perp$	2	perp	$\Downarrow$	2 Downarrow
$\times$	2	times	$\equiv$	2	equiv	$\Updownarrow$	2 Updownarrow
$\star$	1	star	$\asymp$	2	asymp	\	2 backslash
$\propto$	2	propto	$\smile$	1	smile	$\rangle$	2 rangle
$\sqsubseteq$	2	sqsubseteq	$\frown$	1	frown	$\langle$	2 langle

}	2 rbrace	⌋	2 rfloor	‡	2 ddag
{	2 lbrace	⌊	2 lfloor	§	2 S
⌉	2 rceil	√	2 sqrt	¶	2 P
⌈	2 lceil	†	2 dag	◯	2 Orb

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	PDFTEX map file definitions
type-spe	special definitions, like math collections
type-siz	size specifications
type-pre	predefined typescripts, downward compatible with font-*.tex
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 mathtimes typeface.

```
\usetypescript[mathtimes][texnansi]
\switchtobodyfont[mathtimes,11pt]
```

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

```
\starttypescript [mathtimes] [texnansi,ec]

\definetypeface [mathtimes] [rm] [serif] [times]           [default] [encoding=\
\definetypeface [mathtimes] [ss] [sans]  [helvetica]       [default] [rscale=.9,
\definetypeface [mathtimes] [tt] [mono]  [computer-modern] [default] [rscale=1.1
\definetypeface [mathtimes] [mm] [math]  [times]           [default] [encoding=\


\stoptypescript
```

These fonts combine in an acceptable way:

In ConTEXt, 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 €symbols. 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. ConTEXt will complete this name according to the current style. You can now access this symbol with `\symbol [euro]`

	\tf	\bf	\sl	\it	\bs	\bi
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 PDFTEX 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 TEX distributions come with a couple of fonts, most noticeably the Computer Modern Roman typefaces. In order to use a font, TEX 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
`tfm`	TEX specific font metric files that, in many cases, can be generated from `afm` files
`vf`	virtual font files, used for building glyph collections from other ones
`afm`	Adobe font metric files that are nore limited than `tfm` files (especially for math fonts)
`pfm`	Windows specific font metric files, not used by TEX applications
`pfb`	files that contain the outline specification of the glyphs fonts, also called Type 1
`enc`	files with encoding vector specifications
`map`	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.[4] The directory structure normally is as follows:

```
texmf-local / fonts  / tfm    / vendor   / name / *.tfm
                     / afm    / vendor   / name / *.afm
                     / pfm    / vendor   / name / *.pfm
                     / vf     / vendor   / name / *.vf
                     / type1  / vendor   / name / *.pfb
```

[4] If you have installed TETEX or FPTEX (possibly from the TEXlive CDROM) you will have many thousands of font files on your system.

```
/ pdftex / config /                  / *.enc
         / config / encoding         / *.map
                                       *.cfg
```

The `texmf-local` tree normally contains your own fonts, so that you don't have to reinstall them when you reinstall teh main tree. The `pdftex` directory contains the files that PDFTEX 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.

CONTEXT comes with a PERL script `texfont.pl` that you can use to install new fonts. 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.

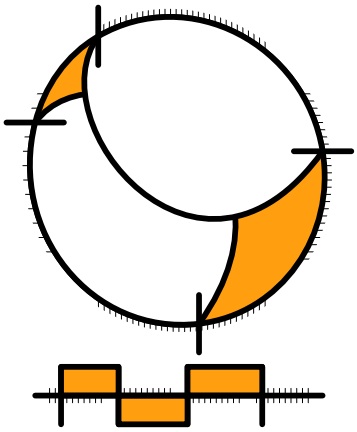
Installing such a font can be handled by this script. For this you need to know (or invent) the name of the font vendor, as well as the name of the font. The full set of command line switches is given below:

switch	**meaning**
`fontroot`	texmf font root (default: `../texmf-local`)
`vendor`	vendor name/directory
`collection`	font collection
`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
`noshow`	don't run tex on texfont.tex

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 --ven=FontFun --col=FirstFont --enc=texnansi --makepath
```

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.



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