Fonts in ConTeXt

Hans Hagen

November 2005

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 provide `\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 rely on a low level mapping mechanism that is responsible for resolving the real font file name and the specific font encoding used.

When TEX users install one of the TEX distributions, like TEX-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, ConTEXt comes with `texfont.pl`, that can install fonts for you. Also, to help you on your way, we provide typescripts for a couple of free fonts.

2 Font 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 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 % [lbr]
```

[lucida] [11.0pt]							
text	script	scriptscript	x	xx	small	big	interlinespace
20.7pt	14.4pt	12pt	17.3pt	14.4pt	17.3pt	20.7pt	not set
17.3pt	12.1pt	8.6pt	13.8pt	10.3pt	13.8pt	20.7pt	not set
14.4pt	10pt	7.2pt	11.5pt	8.6pt	11.5pt	17.2pt	not set
12pt	8.3pt	6pt	9.6pt	7.2pt	9.6pt	14.3pt	not set
11pt	7.6pt	5.5pt	8.8pt	6.6pt	8.8pt	13.1pt	not set
10pt	6.9pt	5pt	8pt	6pt	8pt	11.9pt	not set
9pt	6.2pt	4.5pt	7.2pt	5.4pt	7.2pt	10.7pt	not set
8pt	5.5pt	4pt	6.4pt	4.8pt	6.4pt	9.5pt	not set
7pt	4.8pt	3.5pt	5.6pt	4.2pt	5.6pt	8.3pt	not set
6pt	4.1pt	3pt	4.8pt	3.6pt	4.8pt	7.1pt	not set
5pt	3.4pt	2.5pt	4pt	3pt	4pt	5.9pt	not set
4pt	2.7pt	2pt	3.2pt	2.4pt	3.2pt	4.7pt	not set

Figure 1 The current bodyfont environment.

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

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]
```

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 on 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]
```

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

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

[modern] [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.

[lucida] [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.

[postscript] [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.

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 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 up to 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(s)
`[map] [*] [*]`	map file specifications

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]  [times]     [default] [encoding=texnansi]
```

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

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

```
\definetypeface [whow] [mm]
  [math]  [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.

[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.

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 $t + e + x + t = m + a + t + h$?
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]       % western european languages
\usetypescript[modern][texnansi] % western european languages
\usetypescript[modern][qx]       % eastern european languages
\usetypescript[modern][t5]       % vietnamese users
```

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

```
\usetypescript[postscript][texnansi] % our prefered encoding
\usetypescript[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:

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

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:

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

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

Bold math looks like this:

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

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 the 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:

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

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.

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

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 combined 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 such encodings, each suited for a subset of languages.

encoding	usage	status
texnansi	a combination of TEX and Adobe standard encoding	okay
qx	an encoding that covers most eastern european languages	okay
t5	an encoding dedicated to vietnamese (many (double) accents)	okay
ec (cork)	the prefered encoding of TEX distributions	bugged
8r	a (strange) mixture of encodings	useless
default	the 7 bit ASCII encoding as used by plain TEX	obsolete
il2	iso latin 2 encoding as needed for Czech and Slovak	obsolete
pl0	a native Polish encoding	obsolete

These encodings are font related as is demonstrated in figure 8, 9, 10, and 11. Here we used the `\showfont` command.

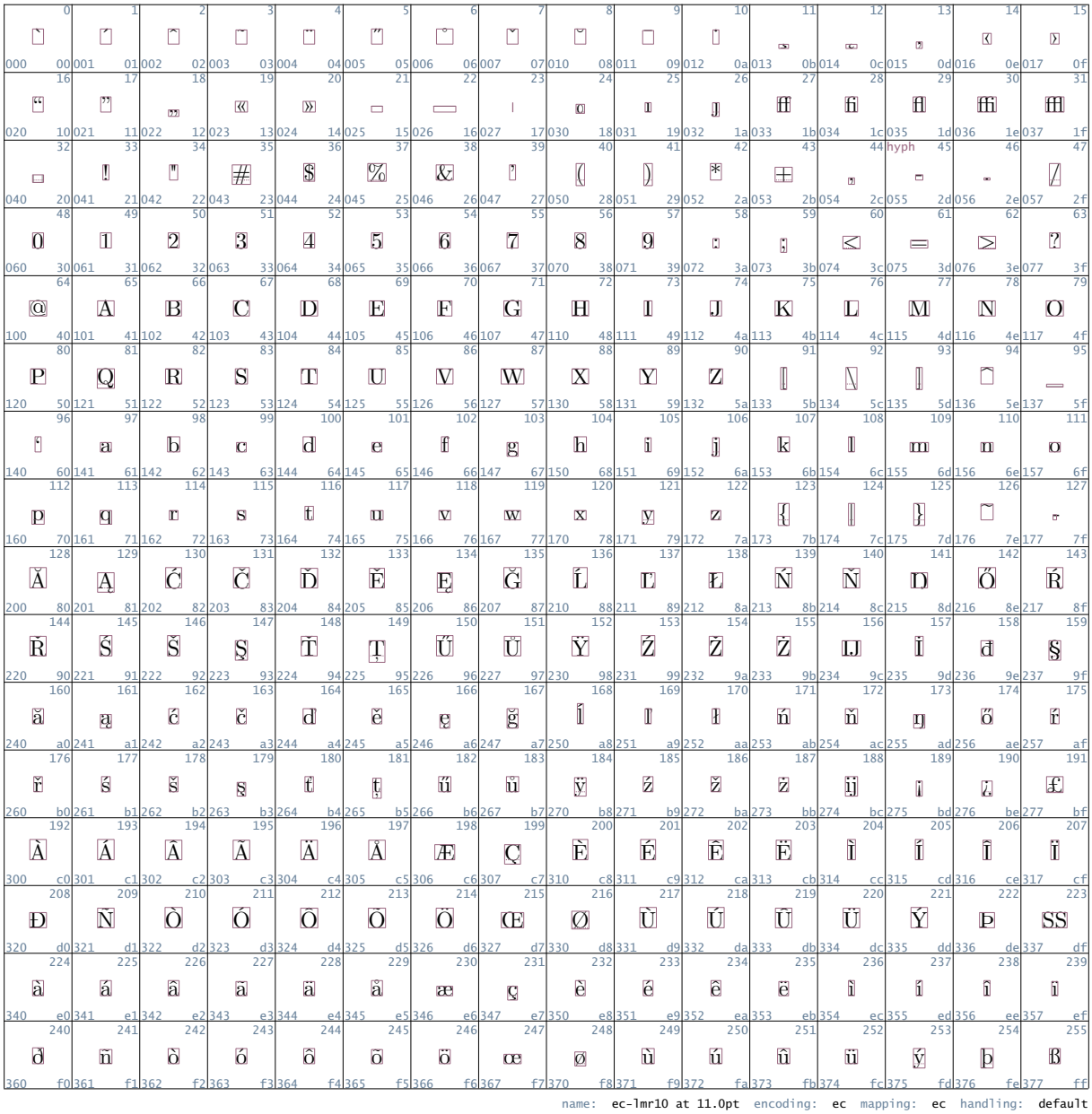


Figure 8 The Latin Modern Roman font in ec encoding.

The situation is even more complicated than it looks, since the font may be virtual, that is, built 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

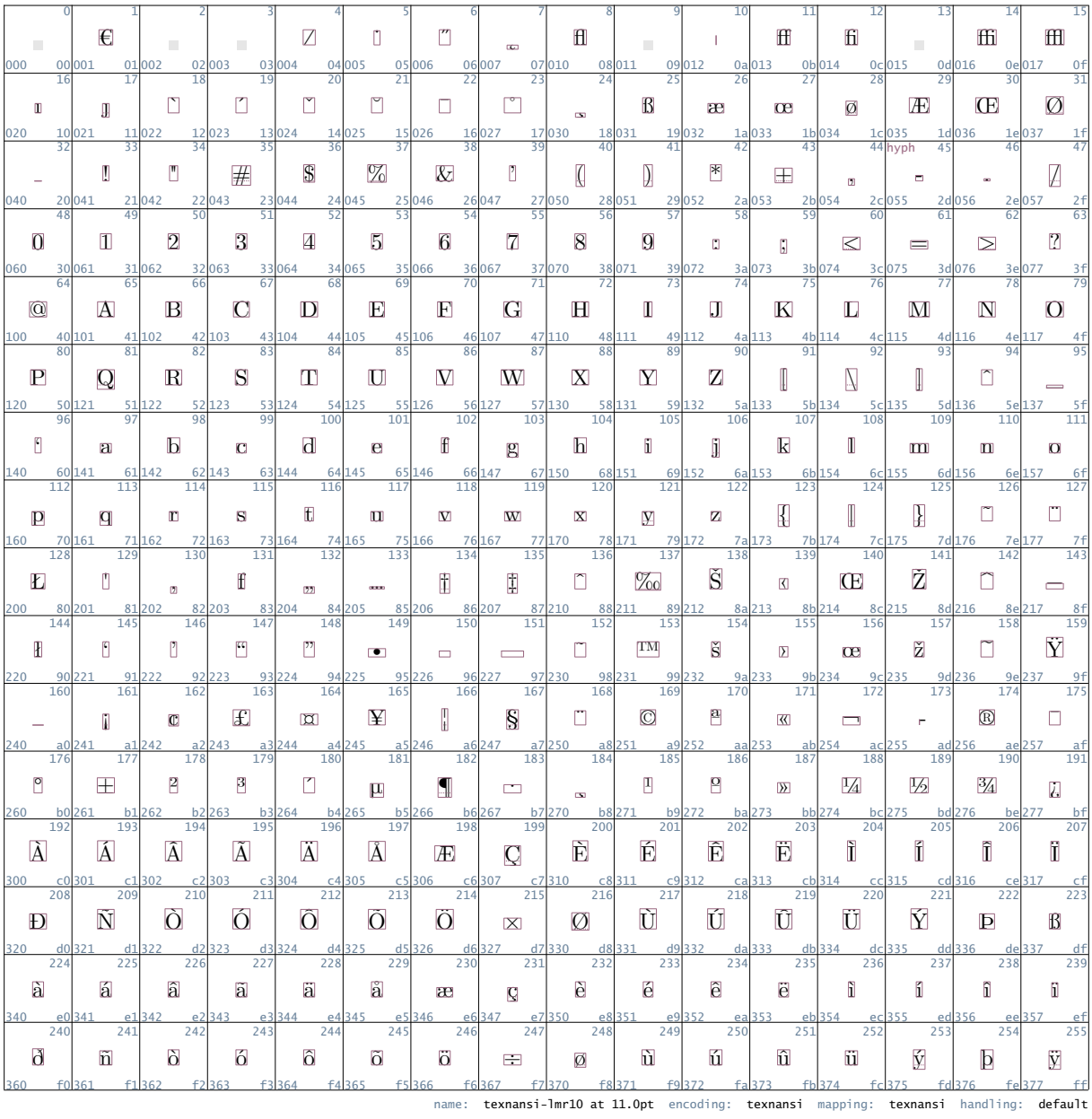


Figure 9 The Latin Modern Roman font in texnansi encoding.

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-

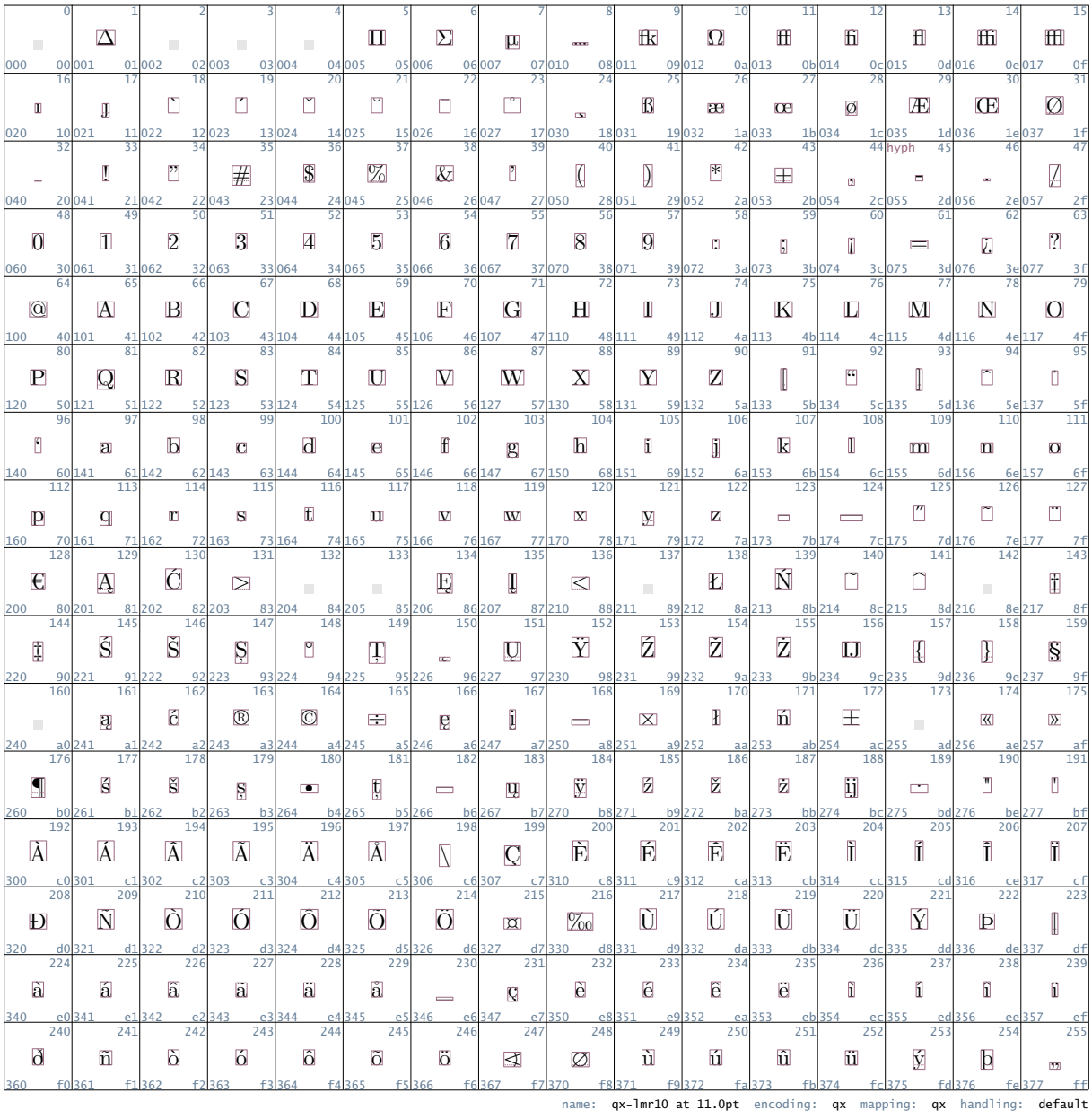


Figure 10 The Latin Modern Roman font in qx encoding.

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 extent 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`:

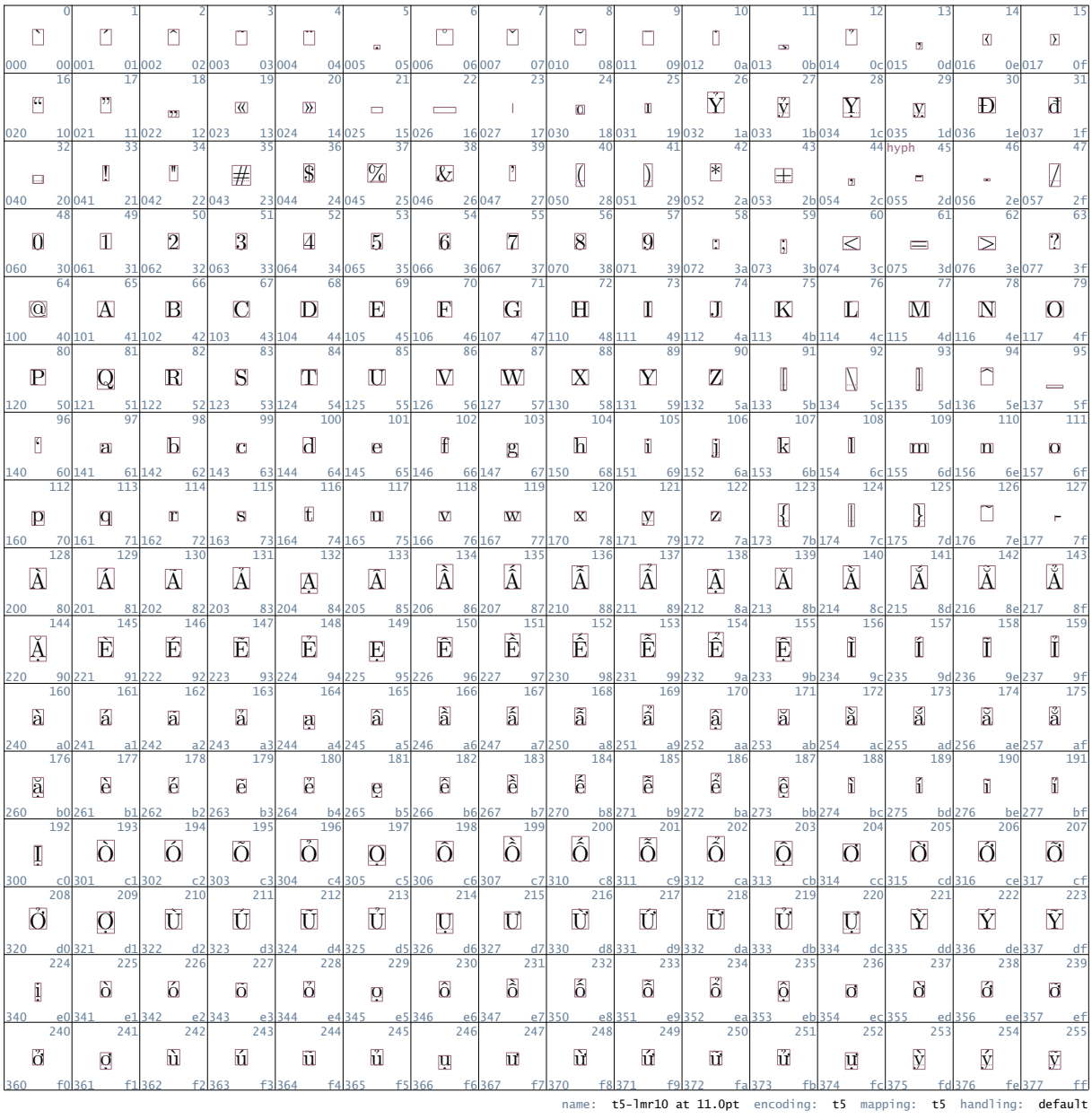


Figure 11 The Latin Modern Roman font in t5 encoding.

texnansi 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	ć	d̀	è	f̀	ǵ	h̀	ì	j̀	ḱ	l̀	m̀	ǹ	ò	p̀	q̀	ŕ	s̀	t̀	ù	v̀	ẁ	x̀	ỳ	ź
	À	B̀	Ć	D̀	È	F̀	Ǵ	H̀	Ì	J̀	Ḱ	L̀	M̀	Ǹ	Ò	P̀	Q̀	Ŕ	S̀	T̀	Ù	V̀	Ẁ	X̀	Ỳ	Ź
\^	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	ȟ	í	j̋	ḱ	ĺ	ḿ	ń	ő	p̋	q̋	ŕ	ś	t̋	ű	v̋	ẃ	x̋	ý	ź
	Á	B̋	Ć	D̋	É	F̋	G̋	Ȟ	Í	J̋	Ḱ	Ĺ	Ḿ	Ń	Ő	P̋	Q̋	Ŕ	Ś	T̋	Ű	V̋	Ẃ	X̋	Ý	Ź
\r	å	b̊	c̊	d̊	e̊	f̊	g̊	h̊	i̊	j̊	k̊	l̊	m̊	n̊	o̊	p̊	q̊	r̊	s̊	t̊	ů	v̊	ẘ	x̊	ẙ	z̊
	Å	B̊	C̊	D̊	E̊	F̊	G̊	H̊	Å	J̊	K̊	L̊	M̊	N̊	O̊	P̊	Q̊	R̊	S̊	T̊	Ů	V̊	W̊	X̊	Y̊	Z̊
\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	c̆	d̆	ĕ	f̆	ğ	h̆	ĭ	j̆	k̆	l̆	m̆	n̆	ŏ	p̆	q̆	r̆	s̆	t̆	ŭ	v̆	w̆	x̆	y̆	z̆
	Ă	B̆	C̆	D̆	Ĕ	F̆	Ğ	H̆	Ĭ	J̆	K̆	L̆	M̆	N̆	Ŏ	P̆	Q̆	R̆	S̆	T̆	Ŭ	V̆	W̆	X̆	Y̆	Z̆
\=	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	j̇	k̇	l̇	ṁ	ṅ	ȯ	ṗ	q̇	ṙ	ṡ	ṫ	u̇	v̇	ẇ	ẋ	ẏ	ż
	Ȧ	Ḃ	Ċ	Ḋ	Ė	Ḟ	Ġ	Ḣ	İ	J̇	K̇	L̇	Ṁ	Ṅ	Ȯ	Ṗ	Q̇	Ṙ	Ṡ	Ṫ	U̇	V̇	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	ọ	ḍ	ẹ	ḟ	ḡ	ḥ	ị	j̣	ḳ	ḷ	ṃ	ṇ	ọ	ṗ	q̣	ṛ	ṣ	ṭ	ụ	ṿ	ẉ	ẋ	ỵ	ẓ
	Ạ	Ḅ	Ọ	Ḍ	Ẹ	Ḟ	Ḡ	Ḥ	Ị	J̣	Ḳ	Ḷ	Ṃ	Ṇ	Ọ	Ṗ	Q̣	Ṛ	Ṣ	Ṭ	Ụ	Ṿ	Ẉ	Ẋ	Ỵ	Ẓ
\k	ą	ḇ	ǫ	ḏ	ę	ḟ	ḡ	ḫ	į	j̨	ḳ	ḷ	ṃ	ṇ	ǫ	ṗ	q̨	ṛ	ṣ	ṭ	ų	ṿ	ẉ	ẋ	ỵ	ẓ
	Ą	Ḇ	Ǫ	Ḏ	Ę	Ḟ	Ḡ	Ḫ	Į	J̨	Ḳ	Ḷ	Ṃ	Ṇ	Ǫ	Ṗ	Q̨	Ṛ	Ṣ	Ṭ	Ų	Ṿ	Ẉ	Ẋ	Ỵ	Ẓ
\c	ą	ḇ	ç	ḑ	ȩ	ḟ	ģ	ḫ	į	j̨	ḳ	ḷ	ṃ	ṇ	ǫ	ṗ	q̨	ṛ	ṣ	ṭ	ų	ṿ	ẉ	ẋ	ỵ	ẓ
	Ą	Ḇ	Ç	Ḑ	Ȩ	Ḟ	Ģ	Ḫ	Į	J̨	Ḳ	Ḷ	Ṃ	Ṇ	Ǫ	Ṗ	Q̨	Ṛ	Ṣ	Ṭ	Ų	Ṿ	Ẉ	Ẋ	Ỵ	Ẓ

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

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

,	textcomma	̨	textogonek	œ	oeligature
.	textperiod	̊	textring	Œ	OEligature
´	textacute	̃	texttilde	ß	ssharp
̣	textbottomdot	ı	dotlessi	SS	Ssharp
̆	textbreve	ȷ	dotlessj	þ	thorn
̌	textcaron	I	dotlessI	Þ	Thorn
̧	textcedilla	J	dotlessJ	ð	eth
̂	textcircumflex	–	endash	Ð	Eth
̈	textdiaeresis	—	emdash	¡	exclamdown
̇	textdotaccent	æ	aeligature	¿	questiondown
`	textgrave	Æ	AEligature	©	copyright
̋	texthungarumlaut	ij	ijligature	®	registered
̄	textmacron	IJ	IJligature	™	trademark

§	sectionmark	µ	textmu	à	agrave
¶	paragraphmark	×	textmultiply	È	Egrave
¼	onequarter	±	textpm	è	egrave
½	onehalf	"	quotedbl	Ì	Igrave
¾	threequarter	„	quotedblbase	ì	igrave
¹	onesuperior	“	quotedblleft	Ò	Ograve
²	twosuperior	”	quotedblright	ò	ograve
³	threesuperior	'	quotesingle	Ù	Ugrave
¢	textcent	,	quotesinglebase	ù	ugrave
¤	textcurrency	‘	quoteleft	Ỳ	Ygrave
$	textdollar	’	quoteright	ỳ	ygrave
	texteuro	‹	guilsingleft	Ã	Atilde
ƒ	textflorin	›	guilsingright	ã	atilde
£	textsterling	«	leftguillemot	Ĩ	Itilde
¥	textyen	»	rightguillemot	ĩ	itilde
ª	ordfeminine	Â	Acircumflex	Ñ	Ntilde
º	ordmasculine	â	acircumflex	ñ	ntilde
%	percent	Ĉ	Ccircumflex	Õ	Otilde
‰	perthousand	ĉ	ccircumflex	õ	otilde
-	softhyphen	Ê	Ecircumflex	Ũ	Utilde
·	periodcentered	ê	ecircumflex	ũ	utilde
°	compoundwordmark	Ĝ	Gcircumflex	Ỹ	Ytilde
^	textasciicircum	ĝ	gcircumflex	ỹ	ytilde
~	textasciitilde	Ĥ	Hcircumflex	Ä	Adiaeresis
/	textslash	ĥ	hcircumflex	ä	adiaeresis
\	textbackslash	Î	Icircumflex	Ë	Ediaeresis
{	textbraceleft	î	icircumflex	ë	ediaeresis
}	textbraceright	Ĵ	Jcircumflex	Ï	Idiaeresis
_	textunderscore	ĵ	jcircumflex	ï	idiaeresis
␣	textvisiblespace	Ô	Ocircumflex	Ö	Odiaeresis
¦	textbrokenbar	ô	ocircumflex	ö	odiaeresis
•	textbullet	Ŝ	Scircumflex	Ü	Udiaeresis
†	textdag	ŝ	scircumflex	ü	udiaeresis
‡	textddag	Û	Ucircumflex	Ÿ	Ydiaeresis
°	textdegree	û	ucircumflex	ÿ	ydiaeresis
÷	textdiv	Ŵ	Wcircumflex	Á	Aacute
…	textellipsis	ŵ	wcircumflex	á	aacute
⁄	textfraction	Ŷ	Ycircumflex	Ć	Cacute
¬	textlognot	ŷ	ycircumflex	ć	cacute
–	textminus	À	Agrave	É	Eacute

é	eacute	ī	imacron	ğ	gbreve
Í	Iacute	Ō	Omacron	Ĭ	Ibreve
í	iacute	ō	omacron	ĭ	ibreve
Ĺ	Lacute	Ū	Umacron	Ŏ	Obreve
Í	lacute	ū	umacron	ŏ	obreve
Ń	Nacute	Ç	Ccedilla	Ŭ	Ubreve
ń	nacute	ç	ccedilla	ŭ	ubreve
Ó	Oacute	Ķ	Kcedilla	Č	Ccaron
ó	oacute	ķ	kcedilla	č	ccaron
Ŕ	Racute	Ļ	Lcedilla	Ď	Dcaron
ŕ	racute	ļ	lcedilla	ď	dcaron
Ś	Sacute	Ņ	Ncedilla	Ě	Ecaron
ś	sacute	ņ	ncedilla	ě	ecaron
Ú	Uacute	Ŗ	Rcedilla	Ľ	Lcaron
ú	uacute	ŗ	rcedilla	Ĭ	lcaron
Ý	Yacute	Ş	Scedilla	Ň	Ncaron
ý	yacute	ş	scedilla	ň	ncaron
Ź	Zacute	Ţ	Tcedilla	Ř	Rcaron
ź	zacute	ţ	tcedilla	ř	rcaron
đ	dstroke	Ő	Ohungarumlaut	Š	Scaron
Đ	Dstroke	ő	ohungarumlaut	š	scaron
H	Hstroke	Ű	Uhungarumlaut	Ť	Tcaron
h	hstroke	ű	uhungarumlaut	ť	tcaron
T	Tstroke	Ą	Aogonek	Ÿ	Ycaron
t	tstroke	ą	aogonek	ÿ	ycaron
Ċ	Cdotaccent	Ę	Eogonek	Ž	Zcaron
ċ	cdotaccent	ę	eogonek	ž	zcaron
Ė	Edotaccent	Į	Iogonek	Ł	Lstroke
ė	edotaccent	į	iogonek	ł	lstroke
Ġ	Gdotaccent	Ų	Uogonek	Ø	Ostroke
ġ	gdotaccent	ų	uogonek	ø	ostroke
İ	Idotaccent	Å	Aring	ä	aumlaut
i	idotaccent	å	aring	ë	eumlaut
Ż	Zdotaccent	Ů	Uring	ï	iumlaut
ż	zdotaccent	ů	uring	ö	oumlaut
Ā	Amacron	Ă	Abreve	ü	uumlaut
ā	amacron	ă	abreve	Ä	Aumlaut
Ē	Emacron	Ĕ	Ebreve	Ë	Eumlaut
ē	emacron	ĕ	ebreve	Ï	Iumlaut
Ī	Imacron	Ğ	Gbreve	Ö	Oumlaut

Ü	Uumlaut	ê	ecircumflexacute	ọ	ohorndotbelow
s’	scommaaccent	ê	ecircumflextilde	Ụ	Uhorndotbelow
S’	Scommaaccent	ê	ecircumflexhook	ụ	uhorndotbelow
t’	tcommaaccent	Ô	Ocircumflexgrave	Ậ	Acircumflexdotbelow
T’	Tcommaaccent	Ô	Ocircumflexacute	ậ	acircumflexdotbelow
Ẽ	Etilde	Ô	Ocircumflextilde	Ệ	Ecircumflexdotbelow
ẽ	etilde	Ô	Ocircumflexhook	ệ	ecircumflexdotbelow
A	Ahook	ô	ocircumflexgrave	Ộ	Ocircumflexdotbelow
a	ahook	ô	ocircumflexacute	ộ	ocircumflexdotbelow
E	Ehook	ô	ocircumflextilde	Ặ	Abrevedotbelow
e	ehook	ô	ocircumflexhook	ặ	abrevedotbelow
I	Ihook	Ă	Abrevegrave	O	Ohorn
i	ihook	Ă	Abreveacute	Ò	Ohorngrave
O	Ohook	Ă	Abrevetilde	Ó	Ohornacute
o	ohook	Ă	Abrevehook	Õ	Ohorntilde
U	Uhook	ă	abrevegrave	O	Ohornhook
u	uhook	ă	abreveacute	o	ohorn
Y	Yhook	ă	abrevetilde	ò	ohorngrave
y	yhook	ă	abrevehook	ó	ohornacute
Â	Acircumflexgrave	Ạ	Adotbelow	õ	ohorntilde
Â	Acircumflexacute	ạ	adotbelow	o	ohornhook
Â	Acircumflextilde	Ẹ	Edotbelow	U	Uhorn
Â	Acircumflexhook	ẹ	edotbelow	Ù	Uhorngrave
â	acircumflexgrave	Ị	Idotbelow	Ú	Uhornacute
â	acircumflexacute	ị	idotbelow	Ũ	Uhorntilde
â	acircumflextilde	Ọ	Odotbelow	U	Uhornhook
â	acircumflexhook	ọ	odotbelow	u	uhorn
Ê	Ecircumflexgrave	Ụ	Udotbelow	ù	uhorngrave
Ê	Ecircumflexacute	ụ	udotbelow	ú	uhornacute
Ê	Ecircumflextilde	Ỵ	Ydotbelow	ũ	uhorntilde
Ê	Ecircumflexhook	ỵ	ydotbelow	u	uhornhook
ê	ecircumflexgrave	Ọ	Ohorndotbelow		

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

language : en (internal code:22)
font : texnansi-lbr at 11.0pt
encoding : texnansi
mapping : not set

handling : not set

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. You can get some insight in what a regime involves by showing it:

```
\showregime[il2]
```

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

If you want to know what regimes are available, you can take a look at the `regi-*.tex` files. A regime that becomes more and more popular is the utf-8 regime. If you want some insight in what vectors provide, you can use commands like:

```
\showunicodevector[001]
```

and

```
\showunicodetable[001]
```

where the last one produces a rather large table.

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's job, 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 phenomenon.

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 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`.

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

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. This mapping vector can be used to enhance other handling vectors with support for accented characters (in most cases á, à, etc. need a treatment similar to a).

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) 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]
```

In addition to these two handlings (hz and protruding) ConTeXt also provides the `noligs` handling (handy when one processes XML), `flexspacing` and `prespacing` (meant for languages like French that need spacing around for instance `:` and `;`). These handlings are somewhat experimental.

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 – lbr

α	1	alpha	K	D	Kappa	♠	2	spadesuit
β	1	beta	Λ	D	Lambda	$\coprod$	3	coprod
γ	1	gamma	M	D	Mu	$\bigvee$	3	bigvee
δ	1	delta	N	D	Nu	$\bigwedge$	3	bigwedge
ϵ	1	epsilon	Ξ	D	Xi	$\biguplus$	3	biguplus
ζ	1	zeta	O	D	Omicron	$\bigcap$	3	bigcap
η	1	eta	Π	D	Pi	$\bigcup$	3	bigcup
θ	1	theta	R	D	Rho	$\int$	3	intop
ι	1	iota	Σ	D	Sigma	$\prod$	3	prod
κ	1	kappa	T	D	Tau	$\sum$	3	sum
λ	1	lambda	Υ	D	Upsilon	$\bigotimes$	3	bigotimes
μ	1	mu	Φ	D	Phi	$\bigoplus$	3	bigoplus
ν	1	nu	X	D	Chi	$\bigodot$	3	bigodot
ξ	1	xi	Ψ	D	Psi	$\oint$	3	ointop
o	1	omicron	Ω	D	Omega	$\bigsqcup$	3	bigsqcup
π	1	pi	$\aleph$	2	aleph	$\int$	2	smallint
ρ	1	rho	$\imath$	1	imath	$\triangleleft$	1	triangleleft
σ	1	sigma	$\jmath$	1	jmath	$\triangleright$	1	triangleright
τ	1	tau	ℓ	1	ell	$\bigtriangleup$	2	bigtriangleup
υ	1	upsilon	$\wp$	1	wp	$\bigtriangledown$	2	bigtriangledown
ϕ	1	phi	$\Re$	2	Re	$\wedge$	2	wedge
χ	1	chi	$\Im$	2	Im	$\vee$	2	vee
ψ	1	psi	∂	1	partial	$\cap$	2	cap
ω	1	omega	∞	2	infty	$\cup$	2	cup
ε	1	varepsilon	$\prime$	2	prime	$\ddagger$	2	ddagger
ϑ	1	vartheta	$\emptyset$	2	emptyset	†	2	dagger
ϖ	1	varpi	∇	2	nabla	$\sqcap$	2	sqcap
ϱ	1	varrho	$\top$	2	top	$\sqcup$	2	sqcup
ς	1	varsigma	$\bot$	2	bot	$\uplus$	2	uplus
φ	1	varphi	$\triangle$	2	triangle	$\amalg$	2	amalg
A	D	Alpha	$\forall$	2	forall	$\diamond$	2	diamond
B	D	Beta	$\exists$	2	exists	$\bullet$	2	bullet
Γ	D	Gamma	$\neg$	2	neg	$\wr$	2	wr
Δ	D	Delta	$\flat$	1	flat	$\div$	2	div
E	D	Epsilon	$\natural$	1	natural	$\odot$	2	odot
Z	D	Zeta	$\sharp$	1	sharp	$\oslash$	2	oslash
H	D	Eta	♣	2	clubsuit	$\otimes$	2	otimes
Θ	D	Theta	◆	2	diamondsuit	$\ominus$	2	ominus
I	D	Iota	♥	2	heartsuit	$\oplus$	2	oplus

∓	2	mp	←	2	leftarrow	↑	2	uparrow
±	2	pm	→	2	rightarrow	↓	2	downarrow
∘	2	circ	↦	2	mapstochar	↕	2	updownarrow
◯	2	bigcirc	∼	2	sim	⇑	2	Uparrow
\	2	setminus	≃	2	simeq	⇓	2	Downarrow
⋅	2	cdot	⊥	2	perp	⇕	2	Updownarrow
∗	2	ast	≡	2	equiv	\	2	backslash
×	2	times	≍	2	asymp	⟩	2	rangle
⋆	1	star	⌣	1	smile	⟨	2	langle
∝	2	propto	⌢	1	frown	}	2	rbrace
⊑	2	sqsubseteq	↼	1	leftharpoonup	{	2	lbrace
⊒	2	sqsupseteq	↽	1	leftharpoondown	⌉	2	rceil
∥	2	parallel	⇀	1	rightharpoonup	⌈	2	lceil
∣	2	mid	⇁	1	rightharpoondown	⌋	2	rfloor
⊣	2	dashv	‘	1	lhook	⌊	2	lfloor
⊢	2	vdash	’	1	rhook	√	2	sqrt
↗	2	nearrow	.	1	ldotp	†	2	dag
↘	2	searrow	·	2	cdotp	‡	2	ddag
↖	2	nwarrow	:	0	colon	§	2	S
↙	2	swarrow	´	0	acute	¶	2	P
⇔	2	Leftrightarrow	`	0	grave	◯	2	Orb
⇐	2	Leftarrow	¨	0	ddot	.	1	mathperiod
⇒	2	Rightarrow	~	0	tilde	.	1	textperiod
≤	2	leq	¯	0	bar	,	1	mathcomma
≥	2	geq	˘	0	breve	,	1	textcomma
≻	2	succ	ˇ	0	check	Γ	0	varGamma
≺	2	prec	^	0	hat	Δ	0	varDelta
≈	2	approx	→	1	vec	Θ	0	varTheta
≽	2	succeq	˙	0	dot	Λ	0	varLambda
≼	2	preceq	~	3	widetilde	Ξ	0	varXi
⊃	2	supset	^	3	widehat	Π	0	varPi
⊂	2	subset	⏜	3	lmoustache	Σ	0	varSigma
⊇	2	supseteq	⏝	3	rmoustache	Υ	0	varUpsilon
⊆	2	subseteq	(	0	lgroup	Φ	0	varPhi
∈	2	in	)	0	rgroup	Ψ	0	varPsi
∋	2	ni	∣	2	arrowvert	Ω	0	varOmega
≫	2	gg	∥	2	Arrowvert	↘	2	internalAnd
≪	2	ll	⏐	3	bracevert	⊡	C	boxdot
/	2	not	∥	2	Vert	⊞	C	boxplus
↔	2	leftrightarrow	∣	2	vert	⊠	C	boxtimes

□ C square	≋ C gtrapprox	◀ C blacktriangleleft
□ C Box	⊸ C multimap	△ C vartriangle
■ C blacksquare	∴ C therefore	△ C triangleup
▪ C centerdot	∵ C because	▲ C blacktriangle
◇ C Diamond	≒ C doteqdot	▽ C triangledown
◇ C lozenge	≑ C Doteq	≖ C eqcirc
♦ C blacklozenge	≜ C triangleq	⋚ C lesseqgtr
↻ C circlearrowright	≾ C precsim	⋛ C gtreqless
↺ C circlearrowleft	≲ C lesssim	⋚ C lesseqqgtr
⇌ C rightleftharpoons	≋ C lessapprox	⋛ C gtreqqless
⇋ C leftrightharpoons	⩽ C eqslantless	⇛ C Rrightarrow
⊟ C boxminus	⩾ C eqslantgtr	⇚ C Lleftarrow
⊩ C Vdash	≼ C curlyeqprec	≚ C veebar
⊪ C Vvdash	≽ C curlyeqsucc	⊼ C barwedge
⊨ C vDash	⋞ C preccurlyeq	⩞ C doublebarwedge
↠ C twoheadrightarrow	≦ C leqq	∠ C angle
↞ C twoheadleftarrow	⩽ C leqslant	⍼ C measuredangle
⇇ C leftleftarrows	≶ C lessgtr	⍲ C sphericalangle
⇉ C rightrightarrows	` C backprime	∝ C varpropto
⇈ C upuparrows	■ C dabar@	⌣ C smallsmile
⇊ C downdownarrows	≒ C risingdotseq	⌢ C smallfrown
↾ C upharpoonright	≓ C fallingdotseq	⋐ C Subset
↾ C restriction	⋟ C succcurlyeq	⋑ C Supset
⇂ C downharpoonright	≧ C geqq	⋓ C Cup
↿ C upharpoonleft	⩾ C geqslant	⋓ C doublecup
⇃ C downharpoonleft	≷ C gtrless	⋒ C Cap
↣ C rightarrowtail	⊏ C sqsubset	⋒ C doublecap
↢ C leftarrowtail	⊐ C sqsupset	⅄ C curlywedge
⇆ C leftrightarrows	▷ C vartriangleright	⅄ C curlyvee
⇄ C rightleftarrows	▷ C rhd	⋋ C leftthreetimes
↰ C Lsh	◁ C lhd	⋌ C rightthreetimes
↱ C Rsh	◁ C vartriangleleft	⊆ C subseteqq
⇝ C rightsquigarrow	⊵ C trianglerighteq	⊇ C supseteqq
⇝ C leadsto	⊵ C unrhd	≏ C bumpeq
⇜ C leftrightsquigarrow	⊴ C trianglelefteq	≎ C Bumpeq
↫ C looparrowleft	⊴ C unlhd	⋘ C llless
↬ C looparrowright	⋆ C bigstar	⋘ C lll
≗ C circeq	≬ C between	⋙ C gggtr
≿ C succsim	▼ C blacktriangledown	⋙ C ggg
≳ C gtrsim	▶ C blacktriangleright	⌜ C ulcorner

⌝	C	urcorner	⋩	D	succneqq	⇏	D	nRightarrow
Ⓢ	C	circledS	⪸	D	precnapprox	⇎	D	nLeftrightarrow
⋔	C	pitchfork	⪹	D	succnapprox	↮	D	nleftrightarrow
∔	C	dotplus	⪺	D	lnapprox	⋇	D	divideontimes
↔	C	backsim	⪻	D	gnapprox	S	D	varnothing
≃	C	backsimeq	≁	D	nsim	∄	D	nexists
⌞	C	llcorner	≇	D	ncong	ⅎ	D	Finv
⌟	C	lrcorner	⟋	D	diagup	ↄ	D	Game
∁	C	complement	⟍	D	diagdown	℧	D	mho
■	C	intercal	⫋	D	varsubsetneq	■	D	eth
⊚	C	circledcirc	⫌	D	varsupsetneq	≃	D	eqsim
⊛	C	circledast	⫅̸	D	nsubseteqq	ב	D	beth
⊝	C	circleddash	⫆̸	D	nsupseteqq	ג	D	gimel
≨	D	lvertneqq	⫅	D	subsetneqq	ד	D	daleth
≩	D	gvertneqq	⫆	D	supsetneqq	⩿	D	lessdot
≰	D	nleq	⫋	D	varsubsetneqq	⪀	D	gtrdot
≱	D	ngeq	⫌	D	varsupsetneqq	⋉	D	ltimes
≮	D	nless	⫋	D	subsetneq	⋊	D	rtimes
≯	D	ngtr	⫌	D	supsetneq	∣	D	shortmid
⊀	D	nprec	⫅̸	D	nsubseteq	∥	D	shortparallel
⊁	D	nsucc	⫆̸	D	nsupseteq	⧹	D	smallsetminus
⪇	D	lneqq	∦	D	nparallel	∼	D	thicksim
⪈	D	gneqq	∤	D	nmid	≈	D	thickapprox
⩽̸	D	nleqslant	∤	D	nshortmid	≊	D	approxeq
⩾̸	D	ngeqslant	∦	D	nshortparallel	⪸	D	succapprox
⪇	D	lneq	⊬	D	nvdash	⪷	D	precapprox
⪈	D	gneq	⊭	D	nVdash	↶	D	curvearrowleft
⪵	D	npreceq	⊯	D	nvDash	↷	D	curvearrowright
⪶	D	nsucceq	⊮	D	nVDash	■	D	digamma
⋨	D	precnsim	⋬	D	ntrianglerighteq	ϰ	D	varkappa
⋩	D	succnsim	⋭	D	ntrianglelefteq	𝕜	D	Bbbk
⋦	D	lnsim	⋪	D	ntriangleleft	ℏ	D	hslash
⋧	D	gnsim	⋫	D	ntriangleright	ℏ	D	hbar
≦̸	D	nleqq	↚	D	nleftarrow	■	D	backepsilon
≧̸	D	ngeqq	↛	D	nrightarrow			
⋨	D	precneqq	⇍	D	nLeftarrow			

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	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 times typeface.[4]

```
\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 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 eurosymbols. You can take a look in symb-eur.tex to see how such a glyph is defined.

[4] We will also add support for MathTimes with texnansi encodings.

```
\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 < texnansi.enc < realname.pfb
```

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. We try to keep up with changes in the map file syntax, the names of fonts, encodings, locations in the TeX tree, etc. However, it remains a troublesome area.

It makes sense to take a look at the `cont-sys.rme` file to see what preferences make sense. If you want to speed up the typescript processing, say (in `cont-sys.tex`:

```
\preloadtypescripts
```

If you want to change the default encoding, you should add something:

```
\setupencoding [default=texnansi]
```

If you want the default berry names:

```
\usetypescript [berry] [\defaultencoding]
```

If you run into missing font metrics kind of problems, you may want to add:

```
\usetypescript[adobekb] [\defaultencoding]
```

Beware: if you use an other encoding afterwards, change this into:

```
\usetypescript[berry]   [all]
\usetypescript[adobekb] [all]
```

You can let ConTeXt load the map files for pdfTeX:

```
\autoloadmapfilestrue

The following lines will remove exiting references to map
files and load a few defaults.

\starttyping
\resetmapfiles

\loadmapfile[original-base.map]
\loadmapfile[original-ams-base.map]
\loadmapfile[original-public-lm.map]
```

As said, map files are a delicate matter.

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 more 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.[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 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. 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.

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

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 *.tex afterwards

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 package 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 a copy of the public Iwona Regular):

```
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}
```

[6] there are a couple of more switches described in the manual mtexfonts.

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
extend=factor	stretch the font to the given factor
narrow=factor	shrink the font to the given factor
slant=factor	create a slanted font
caps=factor	replace lowercase characters by small uppercase ones
test	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 you prefered 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 are probably the most complicated aspect of TEX. If you are not sure that you have the latest ConTEXt, 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 ConTEXt. 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 ConTeXt 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 pdfTeX 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 TeX searches for files (we're talking web2c 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={$TEXMFFONTS,$TEXMFPROJECT,$TEXMFLOCAL,!!$TEXMFMAIN}
```

Here we've added a font path, 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:

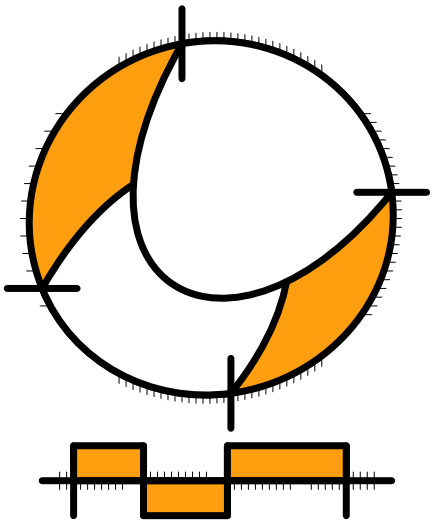
```
\usetypescript [map] [all]
```

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.

16 Remarks

It really makes sense to take a look at the font and type definition files (`font-*.tex` and `type-*.tex`). There are fallbacks defined, as well as generic definitions. Studying styles and manual source code may also teach you a few tricks.



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