Handling fonts in ConTEXt

Willi Egger

Obviously when typesetting electronically, we need to have at least one font available for use as base font. Although there are quite a number of fonts delivered with the distribution, often we need to use a custom font. It is not that complicated to use a third-party font, however one needs to implement it in a structured way in order to make it available to TEX. The following article gives insight into the basic principles of how this can be done in ConTEXt.

Where to place a new font in ConTEXt

In order to make a new font available to ConTEXt, we must make sure that it can find it. The recommended way of doing so is to place the font into the TEX-tree: `.../tex/texmf-fonts/data`. Basically we can just dump it there. If we have many third-party fonts, it might make sense to put the fonts in subdirectories with the foundry name and the font name in a directory thereunder.

ConTEXt operates a file database where all files in the TEX-tree are noted. So after putting a new font into the TEX-tree, this database must be updated:

In a terminal run the following:

```
mtxrun --generate
```

Further ConTEXt maintains an index for the fonts so we have to update this index by running:

```
mtxrun --script font --reload
```

Beyond using third-party fonts, we also can make use of the fonts provided by the operating system.

In order to tell ConTEXt where to find the system fonts, we make use of the environment variable `OSFONTDIR`.

In Windows, we issue the following command on the command line:

```
set OSFONTDIR=c:/windows/fonts
```

Or set the environment variable in the ‘System Properties / Advanced’ tab.

On UNIX-like systems like MacOS and Linux, we have to issue the following command in the terminal:

– Mac:

```
export OSFONTDIR=/Library/fonts:/System/Library/Fonts:~/Library/Fonts
```

– Linux:

```
export OSFONTDIR=~/.fonts:/usr/share/fonts
```

For these UNIX-based systems, we need to add these command line statements to the `.bashrc`, or the corresponding file for the shell we use, to make them permanent.

Again after this, we must regenerate the file database and update the font index with the aforementioned commands.

Finding a font name

ConTEXt comes with a series of tools to identify the available fonts.

For finding font names, we issue the following command in a terminal:

```
mtxrun --script fonts --list --all --pattern="*fontname*"
```

The pattern for the fontname can include wildcards (*).

Example: Lato

Lato is a free font which can be downloaded from *www.latofonts.com/lato-free-fonts/*or *fonts.google.com/specimen/Lato#standard-styles*. The latest version is 2.015.

The Lato font family was designed by Łukasz Dziedzić, Warsaw, from 2010 to 2015.

Running in the terminal:

```
mtxrun --script font --list --file --pattern="*lato*"
```

results in the following list:

family n.	weight	style	width	variant	fontname	filename
lato	black	normal	normal	normal	latoblack	Lato-Black.ttf
lato	black	italic	normal	normal	latoblackitalic	Lato-BlackItalic.ttf
lato	bold	normal	normal	normal	latobold	Lato-Bold.ttf
lato	bold	italic	normal	normal	latobolditalic	Lato-BoldItalic.ttf
lato	light	normal	normal	normal	latohairline	Lato-Hairline.ttf
lato	light	italic	normal	normal	latohairlineitalic	Lato-HairlineItalic.ttf
lato	extrabold	normal	normal	normal	latoheavy	Lato-Heavy.ttf
lato	extrabold	italic	normal	normal	latoheavyitalic	Lato-HeavyItalic.ttf
lato	normal	italic	normal	normal	latoitalic	Lato-Italic.ttf
lato	light	normal	normal	normal	latolight	Lato-Light.ttf
lato	light	italic	normal	normal	latolightitalic	Lato-LightItalic.ttf

lato	medium	normal	normal	normal	latomedium	Lato-Medium.ttf
lato	medium	italic	normal	normal	latomediumitalic	Lato-MediumItalic.ttf
lato	normal	normal	normal	normal	latoregular	Lato-Regular.ttf
lato	semibold	normal	normal	normal	latosemibold	Lato-Semibold.ttf
lato	semibold	italic	normal	normal	latosemibolditalic	Lato-SemiboldItalic.ttf
lato	light	normal	thin	normal	latothin	Lato-Thin.ttf
lato	light	italic	thin	normal	latothinitalic	Lato-ThinItalic.ttf

This list confirms that ConTEXt has found the font family, and now we can start to use it.

Using a font with `\definedfont`

Any font known to ConTEXt can be used with the command `\definedfont`.

```
\definedfont[latomedium*default at 12pt]
```

“Have Fun with Fonts in ConTEXt”

Adding `*default` makes ConTEXt use the default feature set which includes ligatures, kerning, etc.

This approach can be used to setup headings and alike; or in a situation where a piece of text must be typeset in a dedicated font.

Typescripts

Of course, it would be very tedious to type the command `\definedfont[fontname*default at size]` over and over again. For setting up the fonts to be used throughout a document, we need to setup the fonts for the ‘body font environment’. For this purpose there are typescripts.

Typescripts define a font for use throughout a document and enables the usual font switches `\tfa`, `\bf`, `\bi`, etc.

Fonts in the distribution

The ConTEXt distribution comes with some 20 fonts, all of them can be used out-of-the-box. All necessary setups (typescripts) are provided in the distribution and the fonts can simply be loaded using `\setupbodyfont[fontname, style, size]`.

You can find these fonts in the …`/tex/texmf/fonts/opentype/public` and …`/tex/texmf/fonts/truetype/public` directories.

The fonts delivered with the ConTEXt distribution are:

Computer Modern:

name		synonym	remarks
modern	:	modern-base	
modernvariable	:	modern-variable	variable width typewriter font

The TeX Gyre collection of fonts, cross-platform OpenType format:

name		synonym	remarks
pagella	:	palatino	incl. math
termes	:	times	incl. math
heros	:	helvetica	
bonum	:	bookman	incl. math
scholas	:	schoolbook	incl. math
adventor	:	avantgarde	
cursor	:	courier	
chorus	:	chancery	

The DejaVu font family: dejavu and dejavu-condensed.

The IBM Plex family: IBM Plex sans, sans-condensed, serif, mono and sans-Hebrew, Devanagari and Thai.

Six additional fonts (covering serif, sans serif, and monospaced):

name		synonym	style
Gentium	:	gentium	serif
Antykwa Połtawskiego	:	antykwa-poltawskiego	serif
Antykwa Toruńska	:	antykwa-torunska	serif
Kurier	:	kurier	sans
Iwona	:	iwona	sans
Libertinus	:	libertinus	serif

Four additional math fonts:

Euler	:	eulernova
STIX2	:	stix
XITS	:	xits
DejaVu	:	dejavu

Creating a typescript for a third-party font

Typescripts provide a method to map font names onto the basic names inside ConTEXt.

Because these symbolic names are used by the `\definebodyfont` command, we have to use the predefined names for each style and alternative.

	serif	sans	mono	handwriting	calligraphy
tf	Serif	Sans	Mono	Handwriting	Calligraphy
bf	SerifBold	SansBold	MonoBold		
it	SerifItalic	SansItalic	MonoItalic		
sl	SerifSlanted	SansSlanted	MonoSlanted		
bi	SerifBoldItalic	SansBoldItalic	MonoBoldItalic		
bs	SerifBoldSlanted	SansBoldSlanted	MonoBoldSlanted		
sc	SerifCaps	SansCaps	MonoCaps		

A typescript is a start-stop construct with two arguments:

```
\starttypescript[sans][name]
...
\stoptypescript
```

The first argument is the style, and the second is the name of the typescript.

The predefined styles are:

- serif
- sans
- mono
- math
- calligraphy
- handwriting

Font Fallback System

It is possible that a font does not provide all glyphs we would like to use. For this reason ConTEXt has a built-in font fallback system. This makes it possible that a missing glyph is retrieved from a defined fallback font.

For each of the default styles, there is a fallback setup:

- font:fallback:serif
- font:fallback:sans
- font:fallback:mono

The definitions of these fallback setups can be found in the `type-fbk.mkxl` file. For example here are the setups for the sans style:

```
\startsetups [font:fallback:sans]
 \definefontsynonym [Sans] [DefaultFont]
 \definefontsynonym [SansBold] [Sans]
 \definefontsynonym [SansItalic] [Sans]
 \definefontsynonym [SansSlanted][SansItalic]
 \definefontsynonym [SansBoldItalic] [Sans]
 \definefontsynonym [SansBoldSlanted][SansBoldItalic]
 \definefontsynonym [SansCaps] [Sans] [features=smallcaps]
\stopsetups
```

Setting up a Typescript

In order to end up with a useable font within ConTEXt, we need three steps involving three typescripts:

- First typescript: map a symbolic, human-readable name onto the filename by using `\definefontsynonym`
- Second typescript: map the internal basic name onto the human-readable name by using `\definefontsynonym`
- Third typescript: create a typeface by using `\definetypeface`

Example: Lato

First we map the human-readable name onto the font file names:

```
\starttypescript[sans][lato]
 \definefontsynonym[latoregular]   [file:Lato-Regular]
 \definefontsynonym[latobold]      [file:Lato-Bold]
 \definefontsynonym[latoitalic]    [file:Lato-Italic]
 \definefontsynonym[latobolditalic][file:Lato-BoldItalic]
\stoptypescript
```

In a second typescript the internal basic names are linked to these human-readable names:

```
\starttypescript[sans][lato]
 \setups[font:fallback:sans]
 \definefontsynonym[Sans]      [latoregular]   [features=default]
 \definefontsynonym[SansBold]  [latobold]      [features=default]
 \definefontsynonym[SansItalic][latoitalic]    [features=default]
 \definefontsynonym[SansBoldItalic][latobolditalic]
                                               [features=default]
\stoptypescript
```

In this step, we see a third argument to the `\definefontsynonym` command. Here we can invoke features provided by the font.

In the third step the typeface is defined:

```
\starttypescript[MyLato]
 \definetypeface[MyLato][ss][sans][lato][default]
\stoptypescript
```

These three typescripts should be placed in a file with the name starting with `type-imp-` added to the font name e.g. `type-imp-lato.mkxl`.

In an actual document we load this typescript file, tell ConTEXt the name of the typescript to use and set up the body font:

```
\usetypescriptfile[lato]
\usetypescript[MyLato]
```

```
\setupbodyfont[MyLato,ss,12pt]
               {    \getbuffer[sampletext]}
               {\it \getbuffer[sampletext]}
               \bold{\getbuffer[sampletext]}
               {\bi \getbuffer[sampletext]}
```

“Have Fun with Fonts in ConTEXt”

“Have Fun with Fonts in ConTEXt”

“Have Fun with Fonts in ConTEXt”

“Have Fun with Fonts in ConTEXt”

Font features

A lot of information about font features can be found in the font manual.

Modern fonts, such as OTF and TTF, can have many features which can be turned on and off. The list of possible features is long and ConTEXt provides even more features than fonts do. This is due to the fact that the features in ConTEXt are considered to be more of a concept.

Examples of font features are:

liga	general ligatures
tlig	TEX ligatures
trep	TEX ligatures replacements
kern	Kerning information
smcp	Small caps
onum	Oldstyle numbers
tnum	Table numbers
lnum	Line numbers
pnum	Proportional numbers
salt	Stylistic alternates
swash	Swash letters
sub	Subscript
sup	Superscript
ss01	Stylistic set 1

...

ConTEXt can show the features contained in a font:

```
mtxrun --script font --list --info --pattern=lato
```

The output of this run looks as follows:

```
mtx-fonts    gpos features:
mtx-fonts
mtx-fonts    feature script languages
mtx-fonts
mtx-fonts    kern    cyrl    dflt
mtx-fonts            dflt    dflt
mtx-fonts            grek    dflt
mtx-fonts            latn    dflt
mtx-fonts    mark    cyrl    dflt
mtx-fonts            dflt    dflt
mtx-fonts            grek    dflt
mtx-fonts            latn    dflt
mtx-fonts
mtx-fonts    gsub features:
mtx-fonts
mtx-fonts    feature script languages
mtx-fonts
mtx-fonts    calt    cyrl    dflt srb
mtx-fonts            dflt    dflt
mtx-fonts            grek    dflt
mtx-fonts            latn    dflt rom trk
mtx-fonts    case    cyrl    dflt srb
mtx-fonts            dflt    dflt
mtx-fonts            grek    dflt
...
```

This is the list of features provided by the Lato fonts in a compressed form:

gpos features:

kern
mark

gsub features:

calt	numr	ss02
case	onum	ss03
dlig	ordn	ss04
dnom	pnum	subs
frac	salt	sups
liga	sinf	tnum
lnum	ss01	

For the configuration of a font we can define feature sets. These definitions may best be included in the `type-imp-'font-name'` file.

It is advisable to make the new feature set inherit from the default feature set. A font feature definition then looks like this:

```
\definefontfeature
       [mylatofeature]
       [default]
       [onum=yes,
       pnum=yes]
```

```
\definefontfeature
       [f:onum]
       [onum=yes]
```

This new feature set can be used in the typescripts either where the human-readable name is mapped onto the internal basic style name or in the definition of the typeface. And of course it can be used in connection with the `\definedfont` command.

Further there are possibilities to turn on and off features on-the-fly.

There are different variants of these commands:

```
\addfeature{f:tnum}
\feature[+]{f:tnum}
```

Of course, there is also the possibility to subtract or switch off a given font feature by:

```
\subtractfeature{f:tnum}
\feature[-]{f:tnum}
```

Using a comma separated list, we can turn on and off several features simultaneously.

It is also possible to use square brackets instead of curly braces for the (second) argument. This mechanism can be handy for using even-spaced tabular numbers in tables, when the body text uses proportional numbers.

The font manual describes more feature commands (see pages 43 and beyond).

The `\definebodyfont` command

All defined symbolic names use the information that is setup with the `\definebodyfont` command.

```
\starttypescript [sans] [default] [size]
\definebodyfont
 [4pt,5pt,6pt,7pt,8pt,9pt,10pt,11pt,12pt,14.4pt,17.3pt] [rm]
 [tf=Sans sa 1,
 bf=SansfBold sa 1,
 it=SansfItalic sa 1,
 sl=SansfSlanted sa 1,
 bi=SansfBoldItalic sa 1,
 bs=SansfBoldSlanted sa 1,
 sc=SansfCaps sa 1]
\stoptypescript
```

Predefined typescripts also exist for serif and mono. If we need to add a font size permanently, we can easily add such a typescript in the typescript file:

```
\starttypescript[sans][default][size]
 \definebodyfont
  [24pt]
  [tf=Sans sa 1
   bf= SansBold sa 1
   ...]
\stoptypescript
```

The `\definebodyfontenvironment` command

Another set of definitions is contained in the body font environments. These definitions are related to a specific size of the body font. They define ‘script’ and ‘scriptscript’ sizes for math as well as the font switches to ‘x’ and ‘xx’ (`\tfx`, `\tfxx` etc.).

```
\definebodyfontenvironment
  [12pt]
  [text=12pt,
   script=9pt,
   scriptscript=7pt,
   x=10pt,
```

```
    xx=8pt,
    big=12pt,
    small=10pt]
```

The first argument specifies the body font size to which the settings apply. In the second brackets, the relative sizes are defined with units.

When looking at the log file, sometimes there are warnings that a body font size is defined with the hint that this should be done globally. For this, it's not necessary to write the whole `\definebodyfontenvironment`. It is enough to just write:

```
\definebodyfontenvironment[24pt]
```

The `\usebodyfont` command

In documents where we want to use several different fonts, it is good practice to tell ConTEXt in the preamble which fonts we want to use. This makes the system preload the fonts, making switching fonts faster.

Font classes

There is yet another mechanism that makes it possible to use more than one body font in the same document. This can be done by creating classes that assign name-spaces. If we were to say `\setupbodyfont[pagella, 10pt]`, for example, the name-space would be `pagella`. This would ensure that at the moment of the font switch, the whole set of styles and sizes would be available.

Defining a typeface

Now that all elements for the complete typescript file have been setup, we can define the typeface.

A basic typescript can consist of a single `\definetypeface` command for the third-party font.

```
\starttypescript[Mylato]
 \definetypeface[Mylato][sans][ss][lato][default]
\stoptypescript
```

This typescript tells ConTEXt that the name of the typescript is 'Mylato', and that is the name that must be used in the `\setupbodyfont` command. This typeface specifies a sans serif font, which is defined by the typescripts with the name 'lato'.

However, typically typefaces are setup as a group of serif, sans serif, mono, and math fonts.

The complete typescript file

```
% Typescriptfile type-imp-mylato.mkxl

\loadtypescriptfile[dejavu]
\loadtypescriptfile[xits]

\definefontfeature[mylatofeature][default]
       [onum=yes]

\definefontfeature[f:tnum][default]
       [tnum=yes]

\definefontfeature[f:onum]
       [onum=yes]

\definefontfeature[f:no-pnum][default]
       [pnum=no]

\starttypescript[sans][lato]
 \definefontsynonym[latoregular]   [file:Lato-Regular]
 \definefontsynonym[latobold]      [file:Lato-Bold]
 \definefontsynonym[latoitalic]    [file:Lato-Italic]
 \definefontsynonym[latobolditalic][file:Lato-BoldItalic]
\stoptypescript

\starttypescript[sans][lato]
  \setups[font:fallback:sans]
  \definefontsynonym[Sans]      [latoregular]
              [features=mylatofeature]
  \definefontsynonym[SansBold]  [latobold]
              [features=mylatofeature]
  \definefontsynonym[SansItalic][latoitalic]
              [features=mylatofeature]
  \definefontsynonym[SansBoldItalic][latobolditalic]
              [features=mylatofeature]
\stoptypescript

\starttypescript[Mylato]
 \definetypeface [Mylato] [ss]  [sans]  [lato]   [default]
 \definetypeface [Mylato] [rm]  [serif] [dejavu] [default]
 \definetypeface [Mylato] [tt]  [mono]  [dejavu] [default]
 \definetypeface [Mylato] [mm]  [math]  [xits]   [default] [rscale
   =1.2]
\stoptypescript
```

In a document, the following lines are placed in the preamble:

```
\usetypescriptfile[type-imp-mylato.mkxl]
 \usetypescript[Mylato]
 \setupbodyfont[Mylato,ss,10pt]
```

The Lato font in use

The numbers in oldstyle: 1 2 3 4 5 6 7 8 9 0.

The sample file Ward

The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

Inhabitants in Three European Cities

Town	Number of Inhabitants	
Amsterdam	1.149	Million
London	8.982	Million
Warsaw	1.690	Million

Summary

Once ConTEXt can find a font, it can be used by issuing the command `\definedfont[fontname*default at size]`. However, it is better to define the whole environment for a font by means of typescripts. Within those typescripts, the font filenames are mapped onto human-readable fontnames and then these are mapped onto the basic internal names. The typeface is defined by means of another typescript.

It is good practice to define this typescript as well the other styles in order to have a complete set of serif, sans, mono and math. The font setup can be fine-tuned by defining font features which can be invoked by the typescript as a permanent solution. Alternatively, defined font features can also be added and subtracted on-the-fly.

Sometimes one needs to take care of font sizes that are not predefined. By using the command `\definefont`, the range of definitions can easily be extended. In case of such an extension, we also have to take care of the body font environments i.e. to define the font switches for `x` and `xx`, the script and scriptscript sizes for math and the font switches for `\small` and `\big`.

References

- Font manual. Hans Hagen. In the distribution.
- Fonts with ConTEXt. Typescripts explained. Wolfgang Schuster. 2013.
- *wiki.contextgarden.net/Fonts*
- *wiki.contextgarden.net/Typescripts_examples*
- *wiki.contextgarden.net/Style_Alternatives*