

# Languages in Context

explaining luatex and mkiv

Hans Hagen  
PRAGMA ADE



# Contents

## Introduction

### 1 Some basics

1.1	Introduction	5
1.2	Available languages	5
1.3	Switching	7
1.4	Hyphenation	8
1.5	Primitives	9
1.6	Control	9
1.7	Installing	10
1.8	Modes	10

### 2 Hyphenation

2.1	How it works	13
2.2	The last words	15
2.3	Explicit hyphens	16
2.4	Extended patterns	17
2.5	Exceptions	19
2.6	Boundaries	19
2.7	Plug-ins	20
2.8	Blocking ligatures	24
2.9	Special characters	25
2.10	Tracing	26

### 3 Labels

3.1	Introduction	29
3.2	Defining labels	29
3.3	Using labels	29
3.4	Hooks	30

### 4 Numbering

4.1	Introduction	33
4.2	Dates	36

### 5 Typesetting

5.1	Introduction	39
5.2	Spacing	39
5.3	Frequencies	44
5.4	Quotes	48
5.5	Sentences	49
5.6	Local control	50

### 6 Goodies

6.1	Introduction	53
6.2	Spell checking	53
<b>7</b>	<b>Sorting</b>	
7.1	Introduction	57
7.2	How it works	57
<b>A</b>	<b>Appendix</b>	
A.1	The language files	63
A.2	The mtx-patterns script	63
A.3	Installed sorters	63
A.4	Verbose counters	72

# Introduction

This document describes an important property of the T<sub>E</sub>X typesetting system and ConT<sub>E</sub>Xt in particular: the ability to deal with different languages at the same time. With languages we refer to natural languages. So, we're not going to discuss the T<sub>E</sub>X language itself, not MetaPost, nor Lua.

The original application of T<sub>E</sub>X was English that uses the Latin script. The fonts that came with T<sub>E</sub>X were suitable for that usage. When lines became too long they could be hyphenated using so called hyphenation patterns. Due to the implementation for many years there was a close relationship between fonts and hyphenation. Although at some point many more languages and scripts were supported, it was only when the Unicode aware variants showed up that hyphenation and fonts were decoupled. This makes it much more easier to mix languages that use different scripts. Although Greek, Cyrillic, Arabic, Chinese, Japanese, Korean and other languages have been supported for a while using (sometimes dirty) tricks, we now have cleaner implementations.

We can hyphenate words in all languages (and scripts) that have a need for it, that is, split it at the end of a line and add a symbol before and/or after the break. The way words are broken into parts is called hyphenation and so called patterns are used to achieve that goal. The way these patterns are constructed and applied was part of the research related to T<sub>E</sub>X development. The method used is also applied in other programs and is probably one of the few popular ways to deal with hyphenation. There have been ideas about extensions that cover the demands of certain languages but so far nothing better has shown up. In the end T<sub>E</sub>X does a pretty decent job and more advanced tricks don't necessarily lead to better results.

Hyphenation is driven by a language number and that's about it. This means that one cannot claim that T<sub>E</sub>X in its raw form supports languages, other than that it can hyphenate and use fonts that provide the glyphs. It's upto a macro package to wrap this into a mechanism that provides the user an interface. So, when we speak about language support, hyphenation is only one aspect. Labels, like the figure in *figure 1.2* need to adapt to the main document language. When dates are shown they can be language specific. Scientific units and math function names can also be subjected to translation. Registers and other lists have to be sorted according to specific rules. Spacing can differ per language.

In this manual we will cover some of functionality in ConT<sub>E</sub>Xt MkIV that relates to languages (and scripts). This manual is a compliment to other manuals, articles and documentation. Here we mostly focus on the language aspects. Some of the content (or maybe most) might look alien and complex to you. This is because one purpose of this manual is to provide a place to wrap up some aspects of ConT<sub>E</sub>Xt. If you're not interested in that, just stick to the more general manuals that also cover language aspects.

*This document is still under construction. The functionality discussed here will stay and more might show up. Of course there are errors, and they're all mine. The text is not checked for spelling errors. Feel free to let me know what should get added.*

Hans Hagen  
PRAGMA ADE, Hasselt NL  
2013 — 2016

# 1 Some basics

## 1.1 Introduction

In this chapter we will see how we can toggle between languages. A first introduction to patterns will be given. Some details of how to control the hyphenation with specific patterns will be given in a later chapter.

## 1.2 Available languages

When you use the English version of ConT<sub>E</sub>Xt you will default to US English as main language. This means that hyphenation will be US specific, which by the way is different from the rules in GB. All labels that are generated by the system are also in English. Languages can often be accessed by names like `english` or `dutch` although it is quite common to use the short tags like `en` and `nl`. Because we want to be as compatible as possible with MkII, there are quite some synonyms. The following table lists the languages that for which support is built-in.<sup>1</sup>

tag	n	parent	file	synonyms	patterns	characters
af	6		af	afrikaans		
agr	50	gr	agr	ancientgreek		
ala	56	la	la			
ar	26		ar	arabic		
ar-ae	27	ar	ar			
ar-bh	28	ar	ar			
ar-dz	43	ar	ar			
ar-eg	29	ar	ar			
ar-in	30	ar	ar			
ar-iq	40	ar-sy	ar			
ar-jo	41	ar-sy	ar			
ar-kw	31	ar	ar			
ar-lb	42	ar-sy	ar			
ar-ly	32	ar	ar			
ar-ma	44	ar	ar			
ar-om	33	ar	ar			
ar-qa	34	ar	ar			
ar-sa	35	ar	ar			
ar-sd	36	ar	ar			
ar-sy	39	ar	ar			
ar-tn	37	ar	ar			

<sup>1</sup> More languages can be defined. It is up to users to provide the information.

ar-ye	38	ar	ar
ca	53	ca	catalan
cn	46	cn	
cs	16	cs	cz czech
da	4	da	danish
de	3	de	deu german
de-at	11	de	de
de-ch	12	de	de
de-de	10	de	de
deo	9	deo	
en	2	us	eng english
en-gb	13	en	gb uk ukenglish
en-us	14	en	us usenglish
es	52	es	sp spanish
es-es	59	es	es
es-la	60	es	es
fi	22	fi	finish
fr	51	fr	fra french
gr	49	agr	greek
hr	18	hr	croatian
hu	23	hu	hungarian
it	54	it	italian
ja	47	ja	
kr	48	kr	
la	55	la	latin
lt	45	lt	lithuanian
ml	64	ml	malayalam
nb	7	nb	bokmal no norwegian
nl	1	nl	dutch nld
nn	8	nn	nynorsk
pl	15	pl	polish
pt	57	pt	portuguese
pt-br	58	pt	
ro	61	ro	romanian



ru	20	ru	russian
sk	17	sk	slovak
sl	19	sl	slovene
			slovenian
sv	5	sv	swedish
th	63	th	thai
tk	25	tk	turkmen
tr	24	tr	turkish
ua	21	uk	ukrainian
vi	62	vi	vietnamese

---

You can call up such a table with the following commands:

```
\usemodule[languages-system]
```

```
\loadinstalledlanguages
```

```
\showinstalledlanguages
```

Instead you can run `context --global languages-system.mkiv`.

As you can see, many languages have hyphenation patterns but for Japanese, Korean, Chinese as well as Arabic languages they make no sense. The patterns are loaded on demand. The number is the internal number that is used in the engine; a user never has to use that number. Numbers < 1 are used to disable hyphenation. The file tag is used to locate and load a specification. Such files have names like `type lang-nl.lua`.

Some languages share the same hyphenation patterns but can have demands that differ, like labels or quotes. The characters shown in the table are those found in the pattern files. The number of patterns differs a lot between languages. This relates to the systematic behind them. Some languages use word stems, others base their hyphenation on syllables. Some language have inflections which adds to the complexity while others can combine words in ways that demand special care for word boundaries. Of course a low or high number can signal a low quality as well, but most pattern collections are assembled over many years and updated when for instance spelling rules change. I think that we can safely say that most patterns are quite stable and of good quality.

## 1.3 Switching

The document language is set with

```
\mainlanguage[en]
```

but when you want to apply the proper hyphenation rules to an embedded language you can use:

```
\language[en]
```

or just:

```
\en
```

The main language determines what labels show up, how numbering happens, in what way dates get formatted, etc. Normally the `\mainlanguage` command comes before the `\starttext` command.

## 1.4 Hyphenation

In Lua<sub>T</sub><sub>E</sub>X each character that gets typeset not only carries a font id and character code, but also a language number. You can switch language whenever you want and the change will be carried with the characters. Switching within a word doesn't make sense but it is permitted:

```
1 \de incrediblykompliziert      incredi-b-ly-kom-pli-ziert
2 \en incrediblykompliziert      in-cred-i-blykom-pliziert
3 \en incredibly\de kompliziert   in-cred-i-blykom-pli-ziert
4 \en incredibly\de \-kompliziert incredibly-kompliziert
5 \en incredibly\de -kompliziert in-cred-i-bly-kompliziert
```

In the line 4 we have a `\-` between the two words, and in the last line just a `-`. If you look closely you will notice that the snippets can be quite small. If we typeset a word with a 1mm text width we get this:

```
in-
cred-
i-
bly
```

If you are familiar with the details of hyphenation, you know that the number of characters at the end and beginning of a word is controlled by the two variables `\lefthyphenmin` and `\righthyphenmin`. However, these only influence the hyphenation process. What bits and pieces eventually end up on a line is determined by the par builder and there the `\hsize` matters. In practice you will not run into these situations, unless you have extreme long words and a narrow column.

Hyphenation normally is limited to regular characters that make up the alphabet of a language. It is insensitive for capitalization as the following text shows:

This time the mu-si-cal dis-trac-tion while de-vel-op-ing code came from watch-ing youtube per-for-mances of Cory Henry (also known from Snarky Puppy, a conglom-er-ate of ex-cel-lent play-ers). Just search the web for his name with 'Ste-vie Won-der and Michael Jack-son Trib-ute'. There is no key-board he can't play. Another in-ter-est-ing key-board player is Sun Rai (a short name for Rai Thistleth-wayte, just google for 'The Bea-t-les, Come To-gether, Live Pi-ano Acoustic with

Loop Pedal’, or do a combined search with ‘Matt Chamberlain’. Okay, and talking of key-boards, let’s not forget Vika Yermolyeva (vk-goeswild) as she’s one of a kind too on the web. And then there is Jacob Collier, in one word: incredible (or hyphenated the Dutch way incredible, let me repeat that in French incredible).<sup>2</sup>

Of course, names are often short and don’t need to be hyphenated (or the left and right settings prohibit it). Another complication with names is that they can come from another language so we either need to switch language temporarily or we need to add an exception (more about that later).

## 1.5 Primitives

In traditional  $\text{\TeX}$  the language is not a property of a character but is triggered by a signal in the (so called) list. Think of:

```
<language 1>this is <language 2>nederlands<language 1> mixed with english
```

This number is set by the primitive `\language`. Language triggers are injected into the list depending on the value of this number. There is also a `\setlanguage` primitive that can inject triggers without setting the `\language` number. Because in  $\text{\LuaTeX}$  the state is kept with the character you don’t need to worry about the subtle differences here.

In  $\text{\ConTeXt}$  the `\language` and `\setlanguage` commands are overloaded by a more advanced switch macro. You cannot assume that they work as explained in general manuals about  $\text{\TeX}$ . Currently you can still assign a number but that might change. Just consider the language to be an abstraction and don’t mess with this number. Both commands not only change the current language but also do specific initializations when needed.

What characters get involved in hyphenation is historically determined by the so called `\lccode` values. Each character can have such a value which maps an uppercase to a lowercase character. This concept has been extended in  $\varepsilon\text{\TeX}$  where it binds to a pattern set (language). However, in  $\text{\ConTeXt}$  the user never has to worry about such details.

In traditional hyphenation there will not be hyphenated if the sum of `\lefthyphenmin` and `\righthyphenmin` exceeds 62. This limitation is not present in the to be presented  $\text{\Lua}$  variant of this routine as there is no good reason for this limitation other than implementation constraints.

## 1.6 Control

We already mentioned `\lefthyphenmin` and `\righthyphenmin`. These two variables control the area in a word that is subjected to hyphenation. Setting these values is a matter

<sup>2</sup> Get me right, there are of course many more fantastic musicians.

of taste but making them too small can result in bad hyphenation when the patterns are made with the assumptions that certain minima are used. Using a `\lefthyphenmin` of 2 while the patterns are made with a value of 3 in mind is a bad idea.

<code>\lefthyphenmin</code>	<code>\righthyphenmin</code>				
	1	2	3	4	5
1	in-ter-est-ing	in-ter-est-ing	in-ter-est-ing	in-ter-esting	in-ter-esting
2	in-ter-est-ing	in-ter-est-ing	in-ter-est-ing	in-ter-esting	in-ter-esting
3	inter-est-ing	inter-est-ing	inter-est-ing	inter-esting	inter-esting
4	inter-est-ing	inter-est-ing	inter-est-ing	inter-esting	inter-esting
5	inter-est-ing	inter-est-ing	inter-est-ing	inter-esting	inter-esting

When  $\text{\TeX}$  breaks a paragraph into lines it will try do so without hyphenation. When that fails (read: when the badness becomes too high) a next effort will take hyphenation into account.<sup>3</sup> When the badness is still too high, an optional emergency pass can be made but only when the tolerances are set to permit this. In  $\text{Con}\text{\TeX}$ t you can try these settings when you get too many over- or underfull boxes reported on the console.

```
\setupalign[tolerant]
\setupalign[verytolerant]
\setupalign[verytolerant,stretch]
```

Personally I tend to use the last setting, especially in automated flows. After all,  $\text{\TeX}$  will not apply stretch unless it's really needed.

The two `\*hyphenmin` parameters can be set any time and the current value is stored with each character. They can also be set with the language which we will see later.

When  $\text{\TeX}$  hyphenates words it has to decide where a word starts and ends. In traditional  $\text{\TeX}$  the words starts normally at a character that falls within the scope of the hyphenator. It ends at when a box (hlist or vlist) is seen, but also at a rule, discretionary, accent (forget about this in  $\text{Con}\text{\TeX}$ t) or math. An example will be given in the chapter that discussed the Lua alternative.

## 1.7 Installing

todo

## 1.8 Modes

Languages are one of the mechanisms where you can access the current state. There are for instance two (official) macros that contain the current (main) language:

<sup>3</sup> Because in  $\text{Lua}\text{\TeX}$  we always hyphenate there is no real gain in trying not to hyphenate. Because in traditional  $\text{\TeX}$  hyphenation happens on the fly a pass without hyphenating makes more sense.

macro	value
<code>\currentmainlanguage</code>	en
<code>\currentlanguage</code>	en

When we have set `\language[nl]` we get this:

macro	value
<code>\currentmainlanguage</code>	en
<code>\currentlanguage</code>	nl

If you write a style that needs to adapt to a language you can use modes. There are several ways to do this:

```
\language[nl]

\startmode[**en]
  \color[darkred]{main english}
\stopmode

\startmode[*en]
  \color[darkred]{local english}
\stopmode

\startmode[**nl]
  \color[darkblue]{main dutch}
\stopmode

\startmode[*nl]
  \color[darkblue]{local dutch}
\stopmode

\startmodeset
  [*en] {\color[darkgreen]{english set}}
  [*nl] {\color[darkgreen]{dutch set}}
\stopmodeset
```

This typesets:

main english

local dutch

dutch set

When you use setups you can use the following trick:

```
\language[nl]  
  
\startsetups language:en  
  \color[darkorange]{something english}  
\stopsetups  
  
\startsetups language:nl  
  \color[darkorange]{something dutch}  
\stopsetups  
  
\setups[language:\currentlanguage]
```

As expected we get:

something dutch

## 2 Hyphenation

### 2.1 How it works

Proper hyphenation is one of the strong points of T<sub>E</sub>X. Hyphenation in T<sub>E</sub>X is done using so called hyphenation patterns. Making these patterns is an art and most users (including me) happily use whatever is available. Patterns can be created automatically using `patgen` but often manual tweaking is needed too. A pattern looks as follows:

```
patltern
```

This means as much as: you can split the word `pattern` in two pieces, with a hyphen between the two `t`'s. Actually it will also split the word `patterns` because the hyphenation mechanism looks at substrings. When no number between characters in a pattern is given, a zero is assumed. This means as much as *undefined*. An even number inhibits hyphenation, an odd number permits it. The larger the number (weight), the more influence it has. A more restricted pattern is:

```
.patltern.
```

Here the periods set the word boundaries. The pattern dictionary for us english has smaller patterns and the next trace shows how these are applied.

```
. p a t t e r n . . p a t t e r n .
1  1p0a0          1 0 0 0 0 0 0 0
2    0a0t5t0e0    1 0 0 5 0 0 0 0
3      4t3t2      1 0 4 5 2 0 0 0
.1p0a4t5t2e0r0n0. . p a t - t e r n .
```

The effective hyphenation of a word is determined by several factors:

- the current language, each language can have different patterns
- the characters, as some characters might block hyphenation
- the settings of `\lefthyphenmin` and `\righthyphenmin`

A place where a word can be hyphenated is called a discretionary. When T<sub>E</sub>X analyzes a stream, it will inject discretionary nodes into that stream.

```
pat\discretionary{-}{-}{-}tern.
```

In traditional T<sub>E</sub>X hyphenation, ligature building and kerning are tightly interwoven which is quite effective. However, there was also a strong relationship between the current font and hyphenation. This is a side effect of traditional T<sub>E</sub>X having at most 256 characters in a font and the fact that the used character is fact a reference to a slot in

a font. There a character in the input initially ends up as a character node and eventually becomes a glyph node. For instance two characters `fi` can become a ligature glyph representing this combination.

In `LuaTeX` the hyphenation, ligature building and kerning stages are separated and can be overloaded. In `ConTeXt` all three can be replaced by code written in Lua. Because normally hyphenation happens before font logic is applied, there is no relationship with font encoding. I wrote the first Lua version of the hyphenator on a rainy weekend and the result was not that bad so it was presented at the 2014 `ConTeXt` meeting. After some polishing I decided to add this routine to the standard MkIV repertoire which then involved some proper interfacing.

You can enable the Lua variant with the following command:

```
\setuphyphenation[method=traditional]
```

We call this method `traditional` because in principle we can have many more methods and this one is (supposed to be) mostly compatible to the built-in method. This is a global setting. You can switch back with:

```
\setuphyphenation[method=default]
```

In the next sections we will see how we can provide alternatives within the traditional method. These alternatives can be set local and therefore can operate over a limited range of characters.

One complication in interfacing is that `TeX` has grouping (which permits local settings) and we want to limit some of the above functionality using groups. At the same time hyphenation is a paragraph related action so we need to enable the hyphenation related code at a global level (or at least make sure that it gets exercised by forcing a `\par`). That means that the alternative hyphenator has to be quite compatible so that we could just enable it for a whole document. This can have an impact on performance but in practice that can be neglected. In `LuaTeX` the Lua variant is 4 times slower than the built-in one, in `LuajitTeX` it's 3 times slower. But the good news is that the amount of time spent in the hyphenator is relatively small compared to other manipulations and macro expansion. The additional time needed for loading and preparing the patterns into a more Lua specific format can be neglected.

You can check how words get hyphenated using the patterns management script:

```
>mtxrun --script patterns --hyphenate language
```

```
hyphenator      |
hyphenator      | . l a n g u a g e .   . l a n g u a g e .
hyphenator      |   0a2n0               0 0 2 0 0 0 0 0 0
hyphenator      |   2a0n0g0             0 2 2 0 0 0 0 0 0
```



hyphenator		0n1g0u0	0 2 2 1 0 0 0 0 0
hyphenator		0g0u4a0	0 2 2 1 0 4 0 0 0
hyphenator		2g0e0.0	0 2 2 1 0 4 2 0 0
hyphenator		.0l2a2n1g0u4a2g0e0.	. l a n-g u a g e .
hyphenator			
mtx-patterns		us 3 3 : language : lan-guage	

## 2.2 The last words

Mid 2014 we had to upgrade a style for a pdf assembly service: chapters from (technical) school books are combined into arbitrary new books. There are some nasty aspects with this flow: for instance, all section numbers in a chapter are replaced by new numbers and this also involves figure and table prefixes. It boils down to splitting up books, analyzing the typeset content and preparing it for replacements. The structure is described in xml files so that we can generate tables of contents. The reason for not generating from xml sources is that the publisher doesn't have a xml workflow and that books already were available. Also, books from several series are combined and even within a series structure (and rendering) differs.

What has this to do with hyphenation? Writing a style for such a flow always results in a more complex one than estimated and as usual it's in the details. The original style was written in MkII and used some box juggling to achieve reasonable results but in MkIV we can do better.

Each chapter has a title and books get titles and subtitles as well. The titles are typeset each time a new book is composed. This happens within some layout constraints. Think of constraints like these:

- the title goes on top of a shape that doesn't permit much overflow
- there can be very long words (not uncommon in Dutch or German)
- a short word or hyphenated part should not end up on the last line
- the left and right hyphenation minima are at least four

The last requirement is a compromise because in most cases publishers seem to want ragged right not hyphenated rendering (at least in Dutch schoolbooks). The arguments for this are quite weak and probably originate in fear of bad rendering given past experiences. It's this kind of situations that drive the development of the more obscure features that ship with ConT<sub>E</sub>Xt and a (partial) solution for this specific case will be given later.

If you look at thousands of titles and turn these into (small) paragraphs T<sub>E</sub>X does a pretty good job. It's the few exceptions that we need to catch. The next examples demonstrate such an extreme case.









1	a verylongword and then anevenlongerword	a verylongword and then anevenlongerword
2	a verylongword and then anevenlongerword	a verylongword and then anevenlongerword
3	a verylongword and then anevenlongerword	a verylongword and then anevenlongerword
4	a verylongword and then anevenlongerword	a verylongword and then anevenlongerword
5	a verylongword and then anevenlongerword	a verylongword and then anevenlongerword

Of course in practice there need to be some reasonable width and when we pose these limits the longest possible word should fit into the allocated space. In these examples the rule shows the width. In the right columns we see a red colored word and that one will not get hyphenated.

## 2.3 Explicit hyphens

Another special case that we needed to handle were (compound) words with explicit hyphens. Because often data comes from xml files we can not really control the typesetting as in a T<sub>E</sub>X document where the author sees what gets done. So here we need a way to turn these hyphens into proper hyphenation directives and at the same time permit the words to be hyphenated.

1	a very-long-word and then an-even longer-word	a very-long-word and then an-even longer-word
---	---	---

2	 a very-long-word and then an-even longer-word	 a very-long-word and then an-even longer-word
3	 a very-long-word and then an-even longer-word	 a very-long-word and then an-even longer-word
4	 a very-long-word and then an-even longer-word	 a very-long-word and then an-even longer-word
5	 a very-long word and then an-even-longer word	 a very-long word and then an-even-longer word

## 2.4 Extended patterns

As with more opened up mechanisms, in MkIV we can extend functionality. As an example I have implemented the extensions discussed in the article by László Németh in the Proceedings of EuroT<sub>E</sub>X 2006: *Hyphenation in OpenOffice.org* (TUGboat, Volume 27, 2006). The syntax for these extension is somewhat ugly and involves optional offsets and ranges.<sup>4</sup>

```
\registerhyphenationpattern[nl][elë/e=e]
\registerhyphenationpattern[nl][a9atje./a=t,1,3]
\registerhyphenationpattern[en][eighltee/t=t,5,1]
\registerhyphenationpattern[de][c1k/k=k]
\registerhyphenationpattern[de][schif1f/ff=f,5,2]
```

These patterns result in the following hyphenations:

reëel	re-eel
omaatje	oma-tje
eighteen	eight-teen

<sup>4</sup> I'm not sure if there were ever patterns released that used this syntax.

Zucker      Zuk-ker  
 Schiffahrt   Schiffahrt

In a specification, the . indicates a word boundary and numbers indicate the weight of a breakpoint. The optional extended specification comes after the /. The values separated by a = are the pre and post sequences: these end up at the end of the current line and beginning of the next one. The optional numbers are the start position and length. These default to 1 and 2, so in the first example they identify eë (the weights don't count).

There is a pitfall here. When the language already has patterns that for instance prohibit a hyphen between e and type ë, like e2ë, we need to make sure that we give our new one a higher priority, which is why we used a e9ë.

This feature is somewhat experimental and can be improved. Here is a more Lua-ish way of setting such patterns:

```
local registerpattern =
    languages.hyphenators.traditional.registerpattern

registerpattern("nl","e1ë", {
    start  = 1,
    length = 2,
    before = "e",
    after  = "e",
} )

registerpattern("nl","a9atje./a=t,1,3")
```

Just adding extra patterns to an existing set without much testing is not wise. For instance we could add these to the dutch dictionary:

```
\registerhyphenationpattern[nl][e3ë/e=e]
\registerhyphenationpattern[nl][o3ë/o=e]
\registerhyphenationpattern[nl][e3i/e=i]
\registerhyphenationpattern[nl][i3ë/i=e]
\registerhyphenationpattern[nl][a5atje./a=t,1,3]
\registerhyphenationpattern[nl][toma8at5je]
```

That would work oke well for words like

coëfficiënt  
 geïntroduceerd  
 copiëren  
 omaatje  
 tomaatje

However, the last word only goes right because we explicitly added a pattern for it. One reason is that the existing patterns already contain rules to prevent weird hyphenations. The same is true for the accented characters. So, consider these examples and coordinate additional patterns with other users so that errors can be identified.

## 2.5 Exceptions

We have a variant on the  $\text{\TeX}$  primitive `\hyphenation`, the official way to register a specific way to hyphenate a word.

```
\registerhyphenationexception[aaaaa-bbbbbb]
aaaaabbbbbb \par
```

This code is self explaining and results in:

```
aaaaa-
bbbbb
```

There can be multiple hyphens and even multiple words in such a specification:

```
\registerhyphenationexception[aaaaa-bbbbbb cc-ccc-ddd-dd]
aaaaabbbbbb \par
ccccddddd \par
```

We get:

```
aaaaa-
bbbbb
```

```
cc-
ccc-
ddd-
dd
```

## 2.6 Boundaries

A box, rule, math or discretionary will end a word and prohibit hyphenation of that word. Take this example:

```
whatever \par
whatever\hbox{!} \par
\v l whatever\v l \par
whatever$x$ \par
whatever-whatever \par
```

These lines will hyphenate differently and in traditional T<sub>E</sub>X you need to insert penalties and/or glue to get around it. In the Lua variant we can enable that limitation.

```
\definehyphenationfeatures
  [strict]
  [rightedge=tex]
```

Here we show the three variants: traditional T<sub>E</sub>X and Lua with and without strict settings.

default	traditional	traditional strict
what-	what-	what-
ever	ever	ever
whatever!	what-	whatever!
whatever	ever!	whatever
whateveryx	what-	whateveryx
whatever-	ever	what-
whatever	what-	ever
	everyx	what-
	what-	ever
	ever	
	what-	
	ever	

By default ConT<sub>E</sub>Xt is configured to hyphenate words that start with an uppercase character. This behaviour is controlled in T<sub>E</sub>X by the `\uchyph` variable. A positive value will enable this and a negative one disables it.

default 0	default 1	traditional 0	traditional 1
TEXified	TEX-	TEX-	TEX-
	i-	i-	i-
	fied	fied	fied

The Lua variants behaves the same as the built-in implementation (that of course remains the reference).

## 2.7 Plug-ins

The default hyphenator is similar to the built-in one, with a couple of extensions as mentioned. However, you can plug in your own code, given that it does return a proper hyphenation result. One reason for providing this plug is that there are users who want to play with hyphenators based on a different logic. In ConT<sub>E</sub>Xt we already have some methods to deal with languages that (for instance) have no spaces but split on words or syllables. A more tight integration with the hyphenator can have advantages so I will explore these options when there is demand.

A result table indicates where we can break a word. If we have a four character word and can break after the second character, the result looks like this:

```
result = { false, true, false, false }
```

Instead of true we can also have a table that has entries like the extensions discussed in a previous section. Let's give an example of a plug-in.

```
\startluacode
  local subset = {
    a = true,
    e = true,
    i = true,
    o = true,
    u = true,
    y = true,
  }

  languages.hyphenators.traditional.installmethod("test",
    function(dictionary,word,n)
      local t = { }
      for i=1,#word do
        local w = word[i]
        if subset[w] then
          t[i] = {
            before = "<" .. w,
            after  = w .. ">",
            left   = false,
            right  = false,
          }
        else
          t[i] = false
        end
      end
      return t
    end
  )
\stopluacode
```

Here we hyphenate on vowels and surround them by angle brackets when split over lines. This alternative is installed as follows:

```
\definehyphenationfeatures
  [demo]
  [alternative=test]
```

We can now use it as follows:

```
\setuphyphenation[method=traditional]
\sethyphenationfeatures[demo]
```

When applied to one the tufte example we get:

We thrive in information-thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsisize, winnow the wheat from the chaff and separate the sheep from the goats.

A more realistic (but not perfect) example is the following:

```
\startluacode
  local packslashes = false

  local specials = {
    ["!"] = "before", ["?"] = "before",
    ["'"] = "before", [""'] = "before",
    ["/"] = "before", ["\\"] = "before",
    ["#"] = "before",
    ["$"] = "before",
    ["%"] = "before",
    ["&"] = "before",
    ["*"] = "before",
    ["+"] = "before", ["-"] = "before",
    [","] = "before", ["."] = "before",
    [":"] = "before", [";"] = "before",
    ["<"] = "before", [">"] = "before",
    ["="] = "before",
    ["@"] = "before",
    ["("] = "before",
    ["["] = "before",
    ["{"] = "before",
    ["^"] = "before", ["_"] = "before",
    ["`"] = "before",
    ["|"] = "before",
    ["~"] = "before",
    --
    [")"] = "after",
```



```

[""] = "after",
["}"] = "after",
}

languages.hyphenators.traditional.installmethod("url",
function(dictionary,word,n)
    local t = { }
    local p = nil
    for i=1,#word do
        local w = word[i]
        local s = specials[w]
        if s == "after" then
            s = {
                start = 1,
                length = 1,
                after = w,
                left = false,
                right = false,
            }
            specials[w] = s
        elseif s == "before" then
            s = {
                start = 1,
                length = 1,
                before = w,
                left = false,
                right = false,
            }
            specials[w] = s
        end
        if not s then
            s = false
        elseif w == p and w == "/" then
            t[i-1] = false
        end
        t[i] = s
        if packslashes then
            p = w
        end
    end
    return t
end
)
\stopluacode

```

Again we define a plug:

```
\definehyphenationfeatures
  [url]
  [characters=all,
   alternative=url]
```

So, we only break a line after symbols.

**http://www.pragma-ade.nl**

A quick test can look as follows:

```
\starthyphenation[traditional]
  \sethyphenationfeatures[url]
  \tt
  \dontcomplain
  \hsize 1mm
  http://www.pragma-ade.nl
\stopthyphenation
```

Or:

```
http:
/
/
www.
pragma-
ade.nl
```

## 2.8 Blocking ligatures

Yet another predefined feature is the ability to block a ligature. In traditional T<sub>E</sub>X this can be done by putting a {} between the characters, although that effect can get lost when the text is manipulated. The natural way to do this in a Unicode environment is to use the special characters zwj and zwnj.

We use the following example lines:

```
supereffective \blank
superef\zwnj fective
```

and define two featuresets:

```
\definehyphenationfeatures
```

```
[demo-1]
[characters=\zwnj\zwj,
joiners=yes]
```

```
\definehyphenationfeatures
[demo-2]
[joiners=no]
```

We limit the width to 1mm and get:

method=default	method=traditional	method=traditional featureset=demo-1	method=traditional featureset=demo-2
super- ef- fec- tive	super- ef- fec- tive	super- ef- fec- tive	super- ef- fec- tive
supereffec- tive	supereffec- tive	super- ef- fec- tive	supereffec- tive

## 2.9 Special characters

The characters example can be used (to some extend) to do the same as the breakpoints mechanism (compounds).

```
\definehyphenationfeatures
[demo-3]
[characters={()[{}]}]

\starthyphenation[traditional]
\sethyphenationfeatures[demo-3]
\dontcomplain
\hsize 1mm \noindentation
we use (super)special(ized) patterns
\stophyphenation

we
use
(su-
per)spe-
cial(ized)
```



The different hyphenation points are shown with colored bars. Some valid points might not be shown because the font engine can collapse successive discretionary hyphenation points.

Coming back to the use of typesetting 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. (de nl en fr)



## 3 Labels

### 3.1 Introduction

When we started using T<sub>E</sub>X, I naturally started with plain T<sub>E</sub>X. But it didn't take long before we tried L<sup>A</sup>T<sub>E</sub>X. Because our documents were in Dutch one of the first fights with this package was to get rid of the english labels. Because rather soon we decided to cook up an alternative package, a decent label mechanism was one of the first things to show up. And as soon as multiple language typesetting gets into view, such a mechanism becomes one of those language dependent features. In this chapter the basics will be covered.

### 3.2 Defining labels

Before we define a label we need to define a label class. You probably seldom need that but this is how it's done:

```
\definelabelclass [mylabel]
```

There are some classes predefined:

<b>head</b>	(complete) titles like and
<b>label</b>	in-text labels like and Figure
<b>mathlabel</b>	function names like sin and cos
<b>taglabel</b>	labels used for tagging purposed in the backend
<b>btxlabel</b>	labels used in typesetting bibliographic items

The physical units mechanism also uses labels: unit, operator, prefix and suffix. All these labels are defined per language with a fall back on english.

Given that we have defined class `mylabel`, a label itself is set like this:

```
\setupmylabeltext
[en]
[first={<after first>},
second={{before second>},{<after second}}]
```

The first argument (the language) is optional. In the next section we will see how these labels are used. A lot of labels are predefined, in MkIV this happens in the file `lang-txt.lua`. There is no need to adapt this file as you can always add labels run time.

### 3.3 Using labels

How a label is called depends on the way it needs to be used. In any case the main language set determines the language of the label. So, when in an Dutch text we temporary switch to German, the Dutch labels are used.

command	first	second
<code>\leftmylabeltext{tag}</code>	<after first	before second>
<code>\rightmylabeltext{tag}</code>		<after second
<code>\mylabeltext{tag}</code>	<after first	before second>
<code>\mylabeltexts{tag}{text}</code>	<after firsttext	before second>text<after second

## 3.4 Hooks

Some mechanisms have label support built in, most noticeably sections heads and numbered items, like figure captions.

```
\definehead
  [myhead]
  [subsection]

\setuphead
  [myhead]
  [bodypartlabel=bodypartmyhead]

\setuplabeltext
  [en]
  [bodypartmyhead=My Head: ]

\myhead{Welcome}
```

### My Head: 3.4.1 Welcome

The head text label class can be used as follows:

```
\setupheadtext
  [SomeHead=Just A Title]

\subsection
  [title=\headtext{SomeHead}]
```

### 3.4.2 Just A Title

A label will obey the style settings, as in:

```
\definehead
  [MyFancyHead]
  [subsection]
  [style={\bs\setcharactercasing[Words]}]

\setupheadtext
```



```
[SomeHead=just another title]
```

```
\MyFancyHead
```

```
[title=\headtext{SomeHead}]
```

### ***3.4.3 Just Another Title***



## 4 Numbering

### 4.1 Introduction

Numbering is complex and in ConT<sub>E</sub>Xt it's not easy either. This is because we not only have 1, 2, 3 ... but also sub numbers like 1a, 1b, 1c ... or 1.a, 1.b, 1.c ... There can be many levels, different separators, final symbols. As we're talking languages we only discuss conversion here: the mechanism that turns a number in for instance a letter. It happens that the mapping from a number onto a letter is language dependent. The next lines show how English, Spanish and Slovenian numbers:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z aa ab
a b c d e f g h i j k l m n ñ o p q r s t u v w x y z aa
a b c č d e f g h i j k l m n o p r s š t u v z ž aa ab ac
```

You convert a number into a letter with:

```
\convertnumber{alphabetic}{15}
```

There is also `\unconvertnumber` which does not expand unless typesetting is going on. Normally you don't need to bother about this.

The alphabetic converter adapts to the current main language. When a language has no special alphabet, the regular 26 characters are used.

A converter can also convert to a roman numeral, a language specific ordered list, a day or month, an ordinal string and again there can be a language specific conversion. The general conversion macro takes a conversion name and a number. When a conversion can be set (for instance in an itemized list, or in section numbering) you can use these names. You can define additional converters if needed, as long as the converter can handle a number.

```
\defineconversion [alphabetic] [\alphabeticnumerals]
```

Here `\alphabeticnumerals` is a converter. If you look into the source of ConT<sub>E</sub>Xt you will see that many converters are calling out to Lua, where we have implemented those specific conversions. The following table has long and short names. The short one are historic.

month	\monthlong
month:mnem	\monthshort
character	\character
Character	\Character
characters	\characters

Characters	\Characters
AK	\smallcappedcharacters
KA	\smallcappedcharacters
alphabetic a	\alphabeticnumerals
Alphabetic A	\Alphabeticnumerals
number numbers n	\numbers
Numbers N	\Numbers
mediaeval m	\mediaeval
word words	\verbosenumbers
Word Words	\VerboseNumber
ordinal	\ordinalnumber
Ordinal	\Ordinalnumber
romannumerals i r	\romannumerals
Romannumerals I R	\Romannumerals
o	\oldstylenumerals
O	\oldstylenumerals
or	\oldstyleromannumerals
KR	\smallcappedromannumerals
RK	\smallcappedromannumerals
greek g	\greeknumerals
Greek G	\Greeknumerals
mathgreek	\mathgreek
abjadnumerals	\abjadnumerals
abjadnodotnumerals	\abjadnodotnumerals
abjadnaivenumerals	\abjadnaivenumerals
thainumerals	\thainumerals
devanagarinumerals	\devanagarinumerals
gurmukhinumerals	\gurmukhinumerals
gujaratinumerals	\gujaratinumerals
tibetannumerals	\tibetannumerals
greeknumerals	\greeknumerals
Greeknumerals	\Greeknumerals
arabicnumerals	\arabicnumerals
persiannumerals	\persiannumerals
arabicexnumerals	\arabicexnumerals
arabicdecimals	\arabicdecimals
persiandecimals	\persiandecimals
koreannumerals kr	\koreannumerals
koreanparenthesisnumerals kr-p	\koreanparenthesisnumerals
koreancirclenumerals kr-c	\koreancirclenumerals

chinesenumerals cn	\chinesenumerals
chinesecapnumerals cn-c	\chinesecapnumerals
chineseallnumerals cn-a	\chineseallnumerals
sloveniannumerals	\sloveniannumerals
slovenianNumerals	\slovenianNumerals
spanishnumerals	\spanishnumerals
spanishNumerals	\spanishNumerals

The alphabetic and Alphabetic converters adapt to slovenian and spanish as do their small capped alternatives. There are more general helpers for it too:

```
\languagecharacters{number}
\languageCharacters{number}
```

Also language related is the \continuednumber macro. Here we see an application:

```
1 \continuednumber{1}
1, 2 \continuednumber{2}
1, 2, 3 \continuednumber{3}
```

What renders as:

```
1
1, 2 (continued)
1, 2, 3 (continued)
```

Such a macro is typically used in combination with counters and it just typesets a label text depending on the value being non-zero.

```
\setuplabeltext[en][continued={and so on}]
1, 2, 3 \continuednumber{3}
1, 2, 3 \convertnumber{continued}{3}
```

This gives:

```
1, 2, 3 and so on
1, 2, 3 and so on
```

In the rare case that you want to check if a conversion is defined you can use

```
\doifelseconversiondefined{name}{true}{false}
```

So,

```
\doifelseconversiondefined{characters}{we can convert}{forget about it}
```

Gives:

we can convert

There are also some non language related converters that we mention here for completeness:

set 0: • – \* ▷ ◦ ● ● □ • – \* ▷ ◦ ● ● □ • – \* ▷

set 1: \* \*\* \* \*\* ‡ ‡ ‡ ‡ \* \*\* \* \*\* \* \*\* \* \*\* ‡ ‡ ‡ ‡ \* \*\* \* \*\* \* \*\*

set 2: \* ‡ ‡ \* \*\* ‡ ‡ \* \*\* ‡ ‡ ‡ \* \*\* ‡ ‡ ‡ ‡ \* ‡ ‡ \* \*\* ‡ ‡ \* \*\* ‡ ‡

set 3: \* \*\* \* \*\* ‡ ‡ ‡ ‡ ¶ ¶ ¶ ¶ § § § § \* \*\* \* \*\* \* \*\* \* \*\* ‡ ‡

When a set overruns we start again at the first element.

The ordinal converter produces output like 123<sup>rd</sup> and 654<sup>th</sup>. The corresponding string renderer is `\highordinalstr`.

## 4.2 Dates

Dates are also language dependent. The following macros take a number and return the name of the month or day.

```
\monthlong   October
\monthshort  oct
\MONTH       OCTOBER
\MONTHLONG   OCTOBER
\MONTHSHORT  OCT
\weekday     Thursday
\WEEKDAY     THURSDAY
```

The current date can be typeset with `\currentdate` and a specific date with `\date`, for instance:

```
\currentdate[weekday,day,month,year]
\currentdate[WEEKDAY,day,MONTH,year]
\date[d=12,m=12,y=1998][weekday]
\date[d=12,m=12,y=1998]
```

```
Tuesday 23 February 2016
TUESDAY 23 FEBRUARY 2016
Saturday
December 12, 1998
```

Possible elements of the specification are:

+ ord	ordinal suffix
++ highord	high ordinal suffix
mnem:	mnemonic prefix
Y y year	year 4 digits
yy	year 2 digits
M	month 1 or 2 digits
mm	month 2 digits
D	day 1 or 2 digits
dd	day 2 digits
W	1 digit
month m	language dependent (can be mnemonic)
day d	language dependent
weekday w	language dependent
MONTH	month uppercased
WEEKDAY	weekday uppercased
referral	YYMMDD
space	space
<word>	word





## 5 Typesetting

### 5.1 Introduction

In this chapter we will discuss a few settings and mechanisms that deal with typesetting from the perspective of languages. We will not go into details about the often obscure demands that a language puts on a system like ConT<sub>E</sub>Xt. Often these are rooted in tradition, limitations of past rendering (brushed, written, mechanical or electronic), subjective decisions made by committees, contradicting opinions of typographers, etc. The most we can do is provide the mechanism that make it possible to honour most of these demands and provide a reasonable set of defaults. It's good to mention here that wasting energy on discussing language specific issues only makes sense when a similar amount of energy is spent on getting the rest of the document rendering right. It really makes no sense to whine about a lost (or bad) ligature, or a missed hyphenation point, or a loose paragraph when vertical spacing is sloppy, the use of color messy, the choice of fonts debatable, etc. The worst discussions I ran into are those involving inter-character spacing as way to optimize look, feel and readability while at the same time the choice of fonts and rest of the layout were not that attractive anyway.

### 5.2 Spacing

The look and feel of a paragraph is determined by several factors and language is undeniable one of them. Dutch and German have compound words that can be quite long, English and French use short words, some with only one character, Czech, Polish and many other languages have diacritics.

Interword spacing makes the text lighter, and the more short words there are, the more spacing shows up. Although, if the hyphenation patterns are suboptimal, spaces can stretch which can become annoying. Many uppercase characters (as in German) and accented characters make the text darker.

The user has not much influence on this, apart from rewriting the text. When a narrow column is used, that can be a bit of a challenge, as too narrow columns can just look bad. In the next example we see a sample text (`tufte.tex`) typeset with the align options `normal`.

We thrive in information-thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, inte-

grate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsise, winnow the wheat from the chaff and separate the sheep from the goats.

We see one word sticking into the margin, and there is not much that  $\text{T}_{\text{E}}\text{X}$  can do about it, given its constraints. We can be a bit more tolerant if we also add the option `tolerant`. This sample text is always good for lots of successive hyphenation I must admit that I never make a big deal about that if only because trying to avoid it often gives worse results.

We thrive in information-thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over,

sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsise, winnow the wheat from the chaff and separate the sheep from the goats.

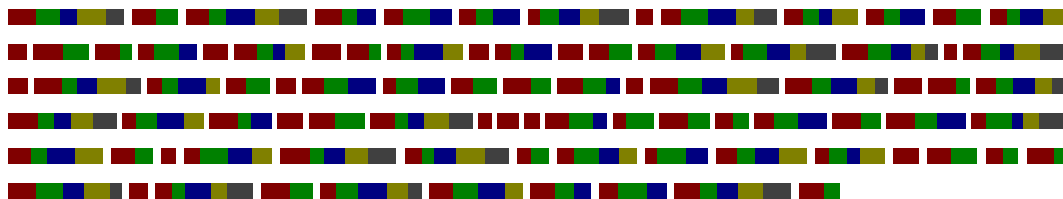
Normally `normal`, `tolerant` is good enough for a document, but if you really want to play safe you can better also permit some stretch.

We thrive in information-thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over,

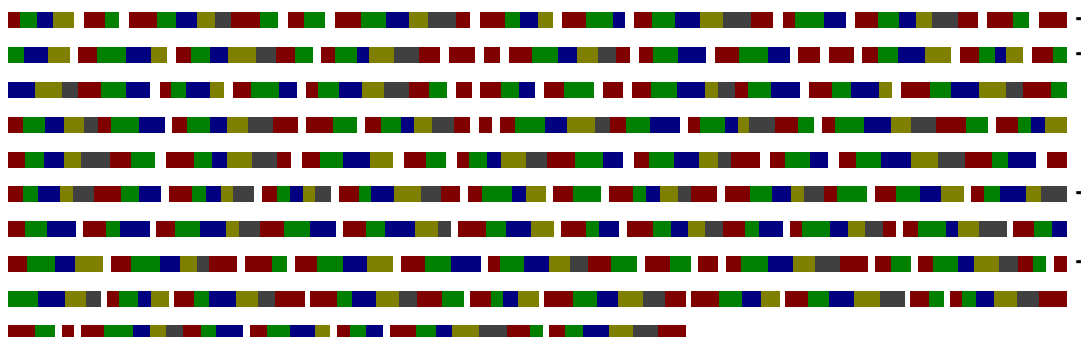
sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsise, winnow the wheat from the chaff and separate the sheep from the goats.

So, `normal`, `tolerant`, `stretch` or `normal`, `verytolerant`, `stretch` gives  $\text{T}_{\text{E}}\text{X}$  enough degrees of freedom to produce good results. When we typeset Dutch documents we always use that alignment setting.

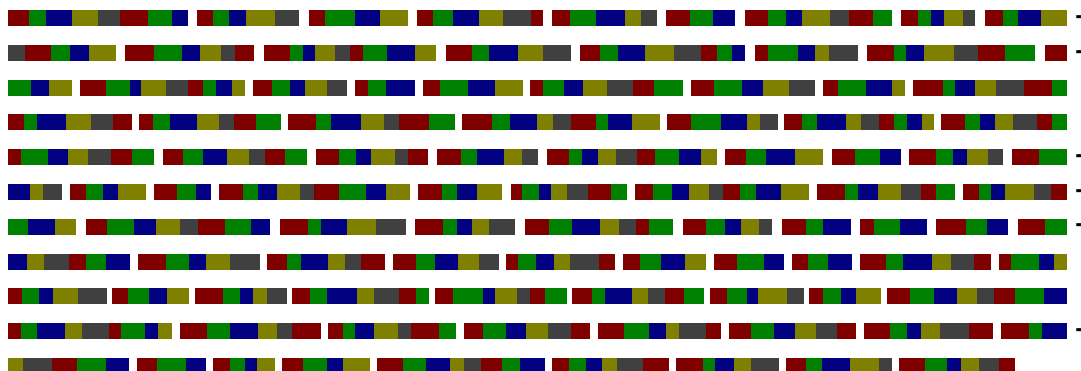
Let's simulate a language with words of an average length. We just randomize the length, permit some hyphenation.



If we assume more longer words we get:



and if we have less short words we get:



Even in the last case  $\text{\TeX}$  can still quite well make a good paragraph, thanks to the large number of possible breakpoints (each color is a unit within a word). If you look careful you will see that the amount of whitespace differs per line.

A space in a text stream becomes so called glue: a horizontal skip with optional stretch and/or shrink. The values come from the font or are derived from spacing related properties of the font. This value can be overloaded by setting the `\spaceskip` register. In the next line we show the font driven spacing of some different fonts:

`x x x x x` `X X`

When we set `\spaceskip` to the 10pt we get:

X X X X

When we use the font related spacing typesetting the Zapf quote gives:

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.

but when we use a fixed `\spaceskip` of 10pt we get the following. This demonstrates that it really makes sense to have some stretch in the skip specification.

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.

The French add spaces before punctuation but no extra space after punctuation. Traditional  $\text{\TeX}$  can only handle additional spacing after characters at the end of a word. The term 'frenchspacing' is used for the feature that disables such extra spacing and `nonfrenchspacing` enables it. The extra space can differ per character and is specified by a factor. That factor (divided by 1000) is applied to the `\xspaceskip`. Spacing before such a final character is supported by  $\text{\ConTeXt}$  but not a generic  $\text{\TeX}$  feature. Frenchspacing looks like:

foo: 1! foo: 2! foo: 3! foo: 4! foo: 5! foo: 6! foo: 7! foo: 8! foo: 9! foo: 10! foo: 11!  
foo: 12! foo: 13! foo: 14! foo: 15! foo: 16! foo: 17! foo: 18! foo: 19! foo: 20! foo: 21!  
foo: 22! foo: 23! foo: 24! foo: 25! foo: 26! foo: 27! foo: 28! foo: 29! foo: 30!

Contrary to:

foo: 1! foo: 2! foo: 3! foo: 4! foo: 5! foo: 6! foo: 7! foo: 8! foo: 9! foo: 10! foo:  
11! foo: 12! foo: 13! foo: 14! foo: 15! foo: 16! foo: 17! foo: 18! foo: 19! foo:  
20! foo: 21! foo: 22! foo: 23! foo: 24! foo: 25! foo: 26! foo: 27! foo: 28! foo:  
29! foo: 30!

The logic is as follows. When the `\spaceskip` is zero, then the spacing after a character is the one related to the font. Otherwise the `\spaceskip` is added. However, when a

character has an `\sfcode` other than 1000, the stretch and shrink component of that glue are multiplied by `\sfcode/1000`. When the `\sfcode` exceeds 2000 the `\xspaceskip` parameter is used. So, what frenchspacing actually does, is resetting those space related codes.

	broad	fixed	packed
.	3000	1000	1050
,	1250	1000	1050
?	3000	1000	1050
!	3000	1000	1050
:	2000	1000	1050
;	1500	1000	1050

The fixed variant is the french spacing as known for ages and discussed in the  $\TeX$  book. You switch the model with:

```
\setupspacing[fixed] % french spacing
```

Additional models can be installed with `\installspacingmethod` and in the source code you can see how we did that for the above.

As mentioned, in  $\TeX$  a space character (U+32) is turned into glue. When you input some text with macros, you sometimes need to get a space into the stream. Take:

I am a  $\TeX$  user.

Here scanning the control sequence  $\TeX$  will gobble the space, so we need to do something:

I am a  $\{\TeX\}$  user, a happy  $\TeX\backslash$  user, yes,  
a proud  $\TeX\{ \}$  user, forever a  $\TeX\backslash\space$  user!

All these cases will inject a space after the logo. The second solution, the  $\backslash$  originally was different because it introduces a space with an explicit `\sfcode` of 1000 (so a factor 1.0) but in Con $\TeX$ t we now just let that be a regular space in text mode and the original (primitive) meaning in math mode.

Another special case is  $\sim$  which is a nobreak space with a fixed width and in some cases (like in tables) a space width the same width as a digit.

fixed	xx xx\ X	xx xx X	xx dr.\ X	xx dr. X
broad	xx xx\ X	xx xx X	xx dr.\ X	xx dr. X
fixed	xx xx X	xx xx X	xx dr. X	xx dr. X
broad	xx xx X	xx xx X	xx dr. X	xx dr. X
fixed	xx xx~X	xx xx X	xx dr.~X	xx dr. X

broad    xx xx~X    **xx xx X**    xx dr.~X    **xx dr. X**

These subtle details probably seldom get noticed. For instance, most languages use the packed variant while English, Turkish and Arabic are configured to use broad. To some extent you can say that the European continent uses the same spacing setup. You can adapt spacing for a language with:

```
\setuplanguage[en][spacing=packed]
```

In addition to a regular space there are many other spacing directives but these always concern fixed width spaces. When possible these spaces travel through the system as Unicode characters which also means that you can use such characters directly.

█	<code>\nobreakspace \nbsp</code>	U+00A0	space
█	<code>\ideographicspace</code>	U+2000	quad/2
█	<code>\ideographichalffillspace</code>	U+2001	quad
█	<code>\twoperemspace</code>	U+2002	quad/2
█	<code>\quad</code>	U+2003	quad (emwidth)
█	<code>\threperemspace</code>	U+2004	quad/3
█	<code>\fourperemspace</code>	U+2005	quad/4
█	<code>\sixperemspace</code>	U+2006	quad/6
█	<code>\figurespace</code>	U+2007	width of 0 (zero)
█	<code>\punctuationspace</code>	U+2008	width of . (period)
█	<code>\breakablethinspace</code>	U+2009	quad/8
█	<code>\hairspace</code>	U+200A	quad/8
█	<code>\zerowidthspace</code>	U+200B	
█	<code>\zerowidthnonjoiner \zwnj</code>	U+200C	
█	<code>\zerowidthjoiner \zwj</code>	U+200D	
█	<code>\narrownobreakspace</code>	U+202F	quad/8
█	<code>\zerowidthnobreakspace</code>	U+FEFF	

The `\nosspacing` macro makes spaces disappear, not even a zero glue will be injected. Of course this macro will only be used grouped and in special cases.

## 5.3 Frequencies

Right from when ConT<sub>E</sub>Xt became multilingual there have been users submitting language specific settings for their language. Some changed over time which indicates that there can be different views, which is not that surprising because many ‘typographic rules’ are just formalizations of ‘this is the way we did it for ages’. I often wonder what rules came from limitations in the systems used to get thing son paper: pens, pensils, letter by letter in wood or lead, line based printing, the size of paper, the reading direction, the quality of ink and paper, and so on. I’ve been present at debates about how

high an accent should be placed on a character, depending on language, referring to some standard well known font that did it this or that way.

The dimensions of a page, the text area, the size and weight of characters, spacing, (the often neglectable or even debatable positive influence of) protrusion into the margin of certain characters, expansion of glyphs to give a better grayness . . . all this can lead to hefty discussions. They don't make a bad looking design or text with all kind of textual elements (not all documents are novels) look better.

The optimal width of a text column is one of these properties can opinions can vary on. Long lines in a small font or short lines in a big one are often not pleasant to read and if you have to turn the page every ten lines you might loose track of the content. But what does it say to have for instance 65 characters on a line. This is quite language dependent. How do spaces count? Anyway, in ConT<sub>E</sub>Xt we have a variable that can help you decide what line length (text width) is acceptable. You can decide yourself what looks better:

322.7436pt The Earth, as a habitat for animal life, is in old age and has a fatal  
 english illness. Several, in fact. It would be happening whether humans had  
 dejavu 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.

327.63428pt The Earth, as a habitat for animal life, is in old age and has a fatal ill-  
 dutch ness. Several, in fact. It would be happening whether humans had ever  
 dejavu 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.

326.45996pt The Earth, as a habitat for animal life, is in old age and has a fatal ill-  
 german ness. Several, in fact. It would be happening whether humans had ever  
 dejavu 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.

322.7436pt The Earth, as a habitat for animal life, is in old age and has a fatal  
 french illness. Several, in fact. It would be happening whether humans had  
 dejavu 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.

282.68692pt The Earth, as a habitat for animal life, is in old age and has a fatal ill-  
 english ness. Several, in fact. It would be happening whether humans had ever  
 pagella 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.

286.82083pt The Earth, as a habitat for animal life, is in old age and has a fatal illness.  
 dutch Several, in fact. It would be happening whether humans had ever evolved  
 pagella 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.

284.73007pt The Earth, as a habitat for animal life, is in old age and has a fatal ill-  
 german ness. Several, in fact. It would be happening whether humans had ever  
 pagella 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.

282.68692pt The Earth, as a habitat for animal life, is in old age and has a fatal ill-  
 french ness. Several, in fact. It would be happening whether humans had ever  
 pagella 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.

The differences are not that large but at least you can play with it. The relevant helpers are:

```
\averagecharwidth
\languagecharwidth{language}
```

These are used like:

```
\hsize=65\averagecharwidth
\hsize=65\languagecharwidth{nl}
```

or when a width is asked for

```
\setupsomething[width=65\averagecharwidth]
\setupsomething[width=65\languagecharwidth{de}]
```

Keep in mind that these are font dependent so you might want to use

```
\freezemeasure[MyWidth][65\averagecharwidth]
```

and then use `\measure{MyWidth}` when needed. The frequencies themselves are stored in tables like `lang-frq-nl.lua`:

```
return {
  language    = "nl",
  source      = "http://www.onzetaal.nl/advies/letterfreq.html",
  frequencies = {
    [0x61] = 7.47, [0x62] = 1.58, [0x63] = 1.24, [0x64] = 5.93, [0x65] = 18.91,
    [0x66] = 0.81, [0x67] = 3.40, [0x68] = 2.38, [0x69] = 6.50, [0x6A] = 1.46,
    [0x6B] = 2.25, [0x6C] = 3.57, [0x6D] = 2.21, [0x6E] = 10.03, [0x6F] = 6.06,
    [0x70] = 1.57, [0x71] = 0.009, [0x72] = 6.41, [0x73] = 3.73, [0x74] = 6.79,
    [0x75] = 1.99, [0x76] = 2.85, [0x77] = 1.52, [0x78] = 0.04, [0x79] = 0.035,
    [0x7A] = 1.39,
  }
}
```



```
}
```

As we only have a few such files, feel free to submit ones that suit your language.

You can show the frequencies in a nice table by loading a module (there is no need to have this in the core):

```
\usemodule[s-languages-frequencies]
```

```
\startcolumns[balance=yes] \en \showfrequencies \stopcolumns
\startcolumns[balance=yes] \de \showfrequencies \stopcolumns
\startcolumns[balance=yes] \nl \showfrequencies \stopcolumns
```

We get three tables:

en: 4.96529pt

U+00061	a	8.040
U+00062	b	1.540
U+00063	c	3.060
U+00064	d	3.990
U+00065	e	12.510
U+00066	f	2.300
U+00067	g	1.960
U+00068	h	5.490
U+00069	i	7.260
U+0006A	j	0.160
U+0006B	k	0.670
U+0006C	l	4.140

U+0006D	m	2.530
U+0006E	n	7.090
U+0006F	o	7.600
U+00070	p	2.000
U+00071	q	0.110
U+00072	r	6.120
U+00073	s	6.540
U+00074	t	9.250
U+00075	u	2.710
U+00076	v	0.990
U+00077	w	1.920
U+00078	x	0.190
U+00079	y	1.730
U+0007A	z	0.090

en: 4.96529pt

U+00061	a	8.040
U+00062	b	1.540
U+00063	c	3.060
U+00064	d	3.990
U+00065	e	12.510
U+00066	f	2.300
U+00067	g	1.960
U+00068	h	5.490
U+00069	i	7.260
U+0006A	j	0.160
U+0006B	k	0.670
U+0006C	l	4.140

U+0006D	m	2.530
U+0006E	n	7.090
U+0006F	o	7.600
U+00070	p	2.000
U+00071	q	0.110
U+00072	r	6.120
U+00073	s	6.540
U+00074	t	9.250
U+00075	u	2.710
U+00076	v	0.990
U+00077	w	1.920
U+00078	x	0.190
U+00079	y	1.730
U+0007A	z	0.090

en: 4.96529pt

U+00061	a	8.040	U+0006D	m	2.530
U+00062	b	1.540	U+0006E	n	7.090
U+00063	c	3.060	U+0006F	o	7.600
U+00064	d	3.990	U+00070	p	2.000
U+00065	e	12.510	U+00071	q	0.110
U+00066	f	2.300	U+00072	r	6.120
U+00067	g	1.960	U+00073	s	6.540
U+00068	h	5.490	U+00074	t	9.250
U+00069	i	7.260	U+00075	u	2.710
U+0006A	j	0.160	U+00076	v	0.990
U+0006B	k	0.670	U+00077	w	1.920
U+0006C	l	4.140	U+00078	x	0.190
			U+00079	y	1.730
			U+0007A	z	0.090

## 5.4 Quotes

The limited support in first generation text editors has made some of these cultural aspects of typesetting disappear or at least it made users sloppy and a new generation forget about them. Quotes are an example and the default rendering on ascii keyboards hasn't helped either. For instance nowadays the Dutch double quotation marks, let's call them lower and upper nine quotes according to their shape, seems to have been replaced by upper double six quotes at the left and upper right double nine quotes on the right. Of course the real names are different.

U+00022	"	quotedbl	quotation mark
U+000AB	«	leftguillemot	left-pointing double angle quotation mark
U+000BB	»	rightguillemot	right-pointing double angle quotation mark
U+02018	'	quoteleft	left single quotation mark
U+02019	'	quoteright	right single quotation mark
U+0201A	,	quotesinglebase	single low-0x0009 quotation mark
U+0201B	`		single high-reversed-0x0009 quotation mark
U+0201C	“	quotedblleft	left double quotation mark
U+0201D	”	quotedblright	right double quotation mark
U+0201E	„	quotedblbase	double low-0x0009 quotation mark
U+0201F	“		double high-reversed-0x0009 quotation mark
U+02039	<	guilsingleleft	single left-pointing angle quotation mark
U+0203A	>	guilsingleright	single right-pointing angle quotation mark
U+02358	⸰		apl functional symbol quote underbar
U+0235E	⸱		apl functional symbol quote quad
U+0275B	‘		heavy single turned comma quotation mark ornament
U+0275C	’		heavy single comma quotation mark ornament
U+0275D	“		heavy double turned comma quotation mark ornament

U+0275E "	heavy double comma quotation mark ornament
U+0275F	heavy low single comma quotation mark ornament
U+02760	heavy low double comma quotation mark ornament
U+0276E ‹	heavy left-pointing angle quotation mark ornament
U+0276F ›	heavy right-pointing angle quotation mark ornament
U+02E42	double low-reversed-9 quotation mark
U+0301D	reversed double prime quotation mark
U+0301E	double prime quotation mark
U+0301F	low double prime quotation mark
U+0A404	yi syllable quot
U+0FF02	fullwidth quotation mark
U+1F676	sans-serif heavy double turned comma quotation mark ornament
U+1F677	sans-serif heavy double comma quotation mark ornament
U+1F678	sans-serif heavy low double comma quotation mark ornament
U+E0022	tag quotation mark

In the language definition file (`lang-def.mkiv`) we use these names; they date from the MkII times:

<code>\lowerleftsingleninequote</code>	<code>\quotesinglebase</code>
<code>\lowerleftdoubleninequote</code>	<code>\quotedblbase</code>
<code>\lowerrightsingleninequote</code>	<code>\quotesinglebase</code>
<code>\lowerrightdoubleninequote</code>	<code>\quotedblbase</code>
<code>\upperleftsingleninequote</code>	<code>\quoteright</code>
<code>\upperleftdoubleninequote</code>	<code>\quotedblright</code>
<code>\upperrightsingleninequote</code>	<code>\quoteright</code>
<code>\upperrightdoubleninequote</code>	<code>\quotedblright</code>
<code>\upperleftsinglesixquote</code>	<code>\quoteleft</code>
<code>\upperleftdoublesixquote</code>	<code>\quotedblleft</code>
<code>\upperrightsinglesixquote</code>	<code>\quoteleft</code>
<code>\upperrightdoublesixquote</code>	<code>\quotedblleft</code>
<code>\leftsubguillemot</code>	<code>\guilsingleleft</code>
<code>\rightsubguillemot</code>	<code>\guilsingleright</code>

In traditional  $\text{T}_{\text{E}}\text{X}$  fonts a `'` and ``` were implemented as ligatures but in  $\text{ConT}_{\text{E}}\text{Xt}$  we never supported that (and we won't). In fact, even using explicit quotes is not advised. When you use `\quotation`, `\quote` and friends the quotes will be used according to the current main language.

## 5.5 Sentences

Another language specific property is sub-sentences (or asides). We just demonstrate some rendering here. Again this is configured in the language definition file.

```
test \aside {test \aside {test} test} test |<|test test|>| test
```

For English, German, Dutch and French this looks as follows. As with quotations a nested instance can render differently.

```
test —test —test— test— test —test test— test
test - test - test - test - test - test test - test
test —test —test— test— test —test test— test
test — test — test — test — test — test test — test
```

## 5.6 Local control

Many ConT<sub>E</sub>Xt commands can be controlled by the `align` parameter that accepts a list of directives. In addition to the justification directives the following ones are used to control the par builder.

option	effect
<code>tolerant</code>	accept suboptimal hyphenation i.e. give less warnings
<code>verytolerant</code>	accept even less optimal hyphenation
<code>stretch</code>	permit stretch between words to satisfy the hyphenation demands
<code>nothyphenated</code>	don't hyphenate at all

These directives are in fact just controllers for the following variables:

variable	effect
<code>\pretolerance</code>	when a paragraph is not hyphenated (first pass) and the result stays within this tolerance T <sub>E</sub> X doesn't try further
<code>\tolerance</code>	when a paragraph is hyphenated (second pass) and the result stays within this tolerance T <sub>E</sub> X doesn't try further
<code>\emergencystretch</code>	permit in a third pass this extra stretch in a line before complaining

You need to keep in mind that the left and right `hyphenmin` variables (per language or locally) also influence the way a paragraph is broken into lines. If you want a paragraph to have more lines than T<sub>E</sub>X want to give it, you can set the `\looseness` variable. Its value is forgotten when the paragraph is typeset so you don't need to reset it yourself. This mechanism will only kick in when there are three passes and a large enough `\emergencystretch` is set.

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.

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.

For this text the results looks quite bad. Personally I never use(d) this command.

There are two commands that can be used to increment or decrement the current values of `\lefthyphenmin` and `\righthyphenmin`:

`\lesshyphens`

`\morehyphens`

These can come in handy in for instance titles.



## 6 Goodies

### 6.1 Introduction

There are some features that will only be used in rare cases. They were often implemented as experiment but found useful enough to keep around.

### 6.2 Spell checking

There are some means to check the spelling of words in your document but get it right: Con<sub>T</sub>E<sub>X</sub>t is not a spell-checker. These features were added in order to be able to do some quick checking of documents written by multiple authors. There are currently three options and we only show a simple examples.

First you need to load word lists. These are either text files with just words separated by spacing.

```
foobar foo-bar foo=bar foo{}{}{}bar foo{}{}{bar}
```

All these words become `foobar` which means that one can use words with discretionary specifications. A text list is loaded with:

```
\loadspellchecklist[en][t:/manuals/luawords-en.txt]
```

Instead you can load a Lua file with words. Here we use the same structure that we use for the spell checker provided for SciTE:

```
return {
    max    = 9,
    min    = 6,
    n      = 2,
    words = {
        ["barfoo"]    = "Barfoo"
        ["foobarred"] = "foobarred",
    }
}
```

We use the same load command (you can also load bytecode files with suffix `luc` this way):

```
\loadspellchecklist[nl][t:/scite/data/context/lexers/data/spell-nl.lua]
```

Usage boils down to enabling the checker. If needed we can add more methods. The first method colors the known and unknown colors. Words shorter than the threshold of 4 will be skipped.

```
\setupspellchecking[state=start,method=1]
\en Is this written right or is this wrong?\par % m -> n error
\nl Is dit goed geschreven of niet?\par
\setupspellchecking[state=stop]
```

Is **this** written right or is **this** wrong?  
Is dit **goed** geschreven of **niet**?

You can change the colors:

```
\definecolor[word:yes] [g=.75]
\definecolor[word:no] [r=.75]
```

The second method doesn't show anything but produces a file `jobname.words`) with used words. The found value of `list` is used as key in the produced table.

```
\setupspellchecking[state=start,method=2,list=found]
\en Is this written right or is this wrong?\par
\nl Is dit goed geschreven of niet?\par
\setupspellchecking[state=stop]
```

Is this written right or is this wrong?  
Is dit goed geschreven of niet?

The produced table is:

```
return {
  ["categories"]={
    ["found"]={
      ["languages"]={
        ["en"]={
          ["list"]={
            ["right"]=1,
            ["this"]=2,
            ["written"]=1,
            ["wrong"]=1,
          },
          ["parent"]="",
          ["patterns"]="us",
          ["tag"]="en",
          ["total"]=5,
          ["unique"]=4,
        },
        ["nl"]={
          ["list"]={
            ["geschreven"]=1,
```



```

    ["goed"]=1,
    ["niet"]=1,
  },
  ["parent"]="",
  ["patterns"]="nl",
  ["tag"]="nl",
  ["total"]=3,
  ["unique"]=3,
},
},
["total"]=8,
},
},
["threshold"]=4,
["total"]=8,
["version"]=1,
}

```

The result can be traced with a module:

```
\usemodule[s-languages-words]
```

```
\showwords
```

This shows up as:

**category: found, language: en, total: 5, unique: 4:** right (1) this (2) written (1) wrong (1)

**category: found, language: nl, total: 3, unique: 3:** geschreven (1) goed (1) niet (1)

The third mechanism colors languages differently. We only defined a few colors:

```

\definecolor[word:en]      [b=.75]
\definecolor[word:de]      [r=.75]
\definecolor[word:nl]      [g=.75]
\definecolor[word:unknown][r=.75,g=.75]

```

but you can of course define a color for your favourite language in a similar way.

```

\setupspellchecking[state=start,method=3]
\en Is this written right or is this wrong?\par
\nl Is dit goed geschreven of niet?\par
\setupspellchecking[state=stop]

```

Is this written right or is this wrong?

Is dit goed geschreven of niet?



## 7 Sorting

### 7.1 Introduction

Sorting is complex, not so much for English, Dutch, German, etc. only texts but there are languages and scripts that are more demanding. There are several complications:

- There can be characters that have accents, like à, á, â, ã, ä . . . that have a base shape a and in an index these often end up close to each other. The order can differ per language.
- There are upper and lowercase words and there can be different expectations to them being mixed or separated.
- Some scripts have characters that are combinations, like Æ, and one might want to see them as one character or two, in which the second one obeys the sorting order. The shape can dominate here.
- Some scripts, like Japanese, are a combination of several scripts and sorting then depends on normalization.
- When there are many glyphs, like in Chinese, the order can depend on the complexity of the glyph and when we're lucky that order is reflected in the numeric character order.

Often the rules are somewhat strict and one can doubt of the same rules would have been imposed if computers had been developed earlier. Given discussions one can doubt if the rules are really consistent or just there because someone (or a group) with influence set the standard (not so much different from grammar). So, if we deal with sorting, we do that in such a way that users can (to some extend) influence the outcome. After all, one important aspect of typesetting and organizing content is that the users gets the feeling of control and a diversion from a standard can be part of that. The reader will often not notice these details. In the next sections we will explore the way sorting is done in ConT<sub>E</sub>Xt. The method evolved over a few decades. In MkII sorting happened between runs and it was just part of the processing of a document that users never really saw in action. Sorting just happened and few users will have noticed that we moved from a Modula program to a Perl script and ended up with a Ruby script. In fact, there is a Lua replacement but it never got tested well because we moved in to MkIV. There all happens inside the engine using Lua. Some principles stayed the same but we are more flexible now.

### 7.2 How it works

How does sorting work out? Take these words:

abracadabra  
 abrăcădăbra  
 àbracádabră  
 ábracadàbra  
 äbrácadabrà

As long as they end up in an order where the reader can find it, we're okay. After all we're pretty good in pattern recognition.

There are probably many ways to implement a sorter but the one we uses is more or less a follow up on the one we had for over a decade and was the result of an evolution based on user demand. It boils down to cleaning up the string in such a way that it can be split into meaningful characters. One can argue that we should use some kd of standardized sorting method but the problem is that we always have to deal with for instance embedded tex commands and mixed content, for instance numbers. And users using the same language can have different opinions about the rules too.

A word (or sequence of words) is split into characters. Because there can be  $\text{\TeX}$  commands in there some cleanup happens beforehand. After that we create several lists with numbers that will be compared when sorting two entries.

We can best demonstrate this with a few examples. As usual an English language example is trivial.

en	abracadabra
ch raw character	a b r a c a d a b r a
uc unicode	x61 x62 x72 x61 x63 x61 x64 x61 x62 x72 x61
zc lowercase	x61 x62 x72 x61 x63 x61 x64 x61 x62 x72 x61
mc lowercase - 1	x61 x62 x72 x61 x63 x61 x64 x61 x62 x72 x61
pc lowercase + 1	x61 x62 x72 x61 x63 x61 x64 x61 x62 x72 x61
zm zero mapping	x02 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02
mm minus mapping	x02 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02
pm plus mapping	x02 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02

When we add an uppercase character we get a slightly different outcome:

en	Abracadabra
ch raw character	A b r a c a d a b r a
uc unicode	x41 x62 x72 x61 x63 x61 x64 x61 x62 x72 x61
zc lowercase	x61 x62 x72 x61 x63 x61 x64 x61 x62 x72 x61
mc lowercase - 1	x60 x62 x72 x61 x63 x61 x64 x61 x62 x72 x61
pc lowercase + 1	x62 x62 x72 x61 x63 x61 x64 x61 x62 x72 x61
zm zero mapping	x02 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02
mm minus mapping	x01 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02
pm plus mapping	x03 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02

Some characters will be split, like æ:

en	æsop
ch raw character	æ s o p
uc unicode	xE6 x73 x6F x70
zc lowercase	xE6 x73 x6F x70
mc lowercase - 1	xE6 x73 x6F x70
pc lowercase + 1	xE6 x73 x6F x70
zm zero mapping	x02 x0A x26 x1E x20
mm minus mapping	x02 x0A x26 x1E x20
pm plus mapping	x02 x0A x26 x1E x20

It gets more complex when language specific demands kick in. Compare an English, German and Austrian split:

en	Abräcàdábra
ch raw character	A b r ä c à d á b r a
uc unicode	x41 x62 x72 xE4 x63 xE0 x64 xE1 x62 x72 x61
zc lowercase	x61 x62 x72 xE4 x63 xE0 x64 xE1 x62 x72 x61
mc lowercase - 1	x60 x62 x72 xE4 x63 xE0 x64 xE1 x62 x72 x61
pc lowercase + 1	x62 x62 x72 xE4 x63 xE0 x64 xE1 x62 x72 x61
zm zero mapping	x02 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02
mm minus mapping	x01 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02
pm plus mapping	x03 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02

de	Abräcàdábra
ch raw character	A b r a e c à d á b r a
uc unicode	x41 x62 x72 x61 x65 x63 xE0 x64 xE1 x62 x72 x61
zc lowercase	x61 x62 x72 x61 x65 x63 xE0 x64 xE1 x62 x72 x61
mc lowercase - 1	x60 x62 x72 x61 x65 x63 xE0 x64 xE1 x62 x72 x61
pc lowercase + 1	x62 x62 x72 x61 x65 x63 xE0 x64 xE1 x62 x72 x61
zm zero mapping	x02 x04 x24 x02 x0A x06 x02 x08 x02 x04 x24 x02
mm minus mapping	x01 x04 x24 x02 x0A x06 x02 x08 x02 x04 x24 x02
pm plus mapping	x03 x04 x24 x02 x0A x06 x02 x08 x02 x04 x24 x02

de-at	Abräcàdábra
ch raw character	A b r ä c à d á b r a
uc unicode	x41 x62 x72 xE4 x63 xE0 x64 xE1 x62 x72 x61
zc lowercase	x61 x62 x72 xE4 x63 xE0 x64 xE1 x62 x72 x61
mc lowercase - 1	x60 x62 x72 xE4 x63 xE0 x64 xE1 x62 x72 x61
pc lowercase + 1	x62 x62 x72 xE4 x63 xE0 x64 xE1 x62 x72 x61
zm zero mapping	x02 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02

mm minus mapping	x01 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02
pm plus mapping	x03 x04 x24 x02 x06 x02 x08 x02 x04 x24 x02

---

The way a character gets replaced, like ä into ae, is defined in `sort-lan.lua` using Lua tables. We will not explain all the obscure details here; most of the work is already done, so users are not bothered by these definitions. And new ones can often be made by copying and adapting an existing one.

The sorting itself is specified by a sequence:

```
default  zc,pc,zm,pm,uc
before   mm,mc,uc
after    pm,mc,uc
first    pc,mm,uc
last     mc,mm,uc
```

The raw character is what we get after the (language specific) replacement has been applied and the unicodes are used when comparing. Lowercasing is done using the Unicode lowercase code, but one can define language specific ones too. The plus and minus variants can be used to force lowercase before or after uppercase. The mapping is based on an alphabet specification so this can differ per language and again we also provide plus and minus values that depend on case. When a character has no case we use shapes instead. For instance, the shape of à is a. Digits are treated special and currently get an offset so that they end up last in the sort order.

```
ぱあ \jindex{ぱあ}
ぱー \jindex{ぱー}
ぱぁ \jindex{ぱぁ}
```

This three entry index should be sorted in the order: `ぱー` `ぱぁ` `ぱあ`.

```
ぱ
ぱあ 60
ぱぁ 60
ぱー 60
```

```
ぱ
ぱあ 60
ぱぁ 60
ぱー 60
```

---

jp	ぱあ
ch raw character	ぱあ
uc unicode	x3071 x3042
zc lowercase	x3071 x3042
mc lowercase - 1	x3071 x3042
pc lowercase + 1	x3071 x3042

---

zm	zero mapping	x34 x02
mm	minus mapping	x34 x02
pm	plus mapping	x34 x02

---

jp	ぱー
----	----

---

ch	raw character	ぱー
uc	unicode	x3071 x30FC
zc	lowercase	x3071 x30FC
mc	lowercase - 1	x3071 x30FC
pc	lowercase + 1	x3071 x30FC
zm	zero mapping	x34 x315C
mm	minus mapping	x34 x315C
pm	plus mapping	x34 x315C

---

jp	ぱぁ
----	----

---

ch	raw character	ぱぁ
uc	unicode	x3071 x3042
zc	lowercase	x3071 x3042
mc	lowercase - 1	x3071 x3042
pc	lowercase + 1	x3071 x3042
zm	zero mapping	x34 x02
mm	minus mapping	x34 x02
pm	plus mapping	x34 x02

---

*To be continued!*





# A Appendix

## A.1 The language files

Todo.

## A.2 The mtx-patterns script

Todo.

## A.3 Installed sorters

```
\usemodule[s-languages-sorting]
```

```
\showinstalledsorting
```

```

language      DIN 5007-1
parent        default
method        mm,mc,uc
replacements  none
order         a b c d e f g h i j k l m n o p q r s t u v w x y z
entries       a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o
                p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z

```

```

language      DIN 5007-2
parent        default
method        mm,mc,uc
replacements  ä=ae Ä=Ae ö=oe Ö=Oe ü=ue Ü=Ue
order         a b c d e f g h i j k l m n o p q r s t u v w x y z
entries       a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o
                p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z

```

```

language      Duden
parent        default
method        mm,mc,uc
replacements  ß=s
order         a b c d e f g h i j k l m n o p q r s t u v w x y z
entries       a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o
                p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z

```

```

language      be
parent        default

```

**method** mm,mc,uc  
**replacements** none  
**order** а б в г д е ё ж з і й к л м н о п р с т у ў ф х ц ч ш ь э ю я  
**entries** а=а б=б в=в г=г д=д е=е ж=ж з=з й=й к=к л=л м=м н=н о=о  
 п=п р=р с=с т=т у=у ф=ф х=х ц=ц ч=ч ш=ш ь=ь э=э ю=ю  
 я=я ё=е і=і ў=ў

**language** bg  
**parent** default  
**method** mm,mc,uc  
**replacements** none  
**order** а б в г д е ж з и й к а л а м н о п р с т у ф х ц ч ш щ ь ю я  
**entries** а=а а=а б=б в=в г=г д=д е=е ж=ж з=з и=и й=й к=к л=л м=м  
 н=н о=о п=п р=р с=с т=т у=у ф=ф х=х ц=ц ч=ч ш=ш щ=щ ь=ь  
 ю=ю я=я

**language** cs  
**parent** cz  
**method** mm,mc,uc  
**replacements** ch=0x10001 Ch=0x10001 CH=0x10001  
**order** а á б с ċ d' é é ě f g h 0x10001 i í j k l m n ň ó p q r ř s š t ť u ú û v  
 w x y ý z ž  
**entries** а=а б=б с=с d=d е=е f=f g=g h=h i=i j=j k=k l=l м=м н=н о=о  
 п=п q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z á=a é=e í=i ó=o  
 ú=u ý=y ċ=ċ d'=d ě=e ň=n ř=ř š=s š=š t=ť ť=ť ú=u ž=ž 0x10001=ch

**language** cu  
**parent** default  
**method** mm,mc,uc  
**replacements** ou=0x10001 OY=0x1000B  
**order** а б в г д е ж s 0x0A643 з 0x0A641 и і ħ к л м н о п р с т у 0x00479  
 0x0A64B 0x10001 ф х 0x00461 0x0047F 0x0047D 0x0A64D ц ч ш щ ь ы  
 ы ь ъ ю ѧ ѧ 0x00467 0x00469 ж ѡ 0x0046F Ѱ ө v ѱ  
**entries** а=а б=б в=в г=г д=д ж=ж з=з и=и к=к л=л м=м н=н о=о п=п  
 р=р с=с т=т у=у ф=ф х=х ц=ц ч=ч ш=ш щ=щ ь=ь ы=ы ъ=ъ ю=ю  
 е=е s=s і=и ħ=ħ 0x00461=0x00461 ѧ=ѧ ѧ=ѧ 0x00467=0x00467  
 0x00469=0x00469 ж=ж ѡ=ѡ 0x0046F=0x0046F Ѱ=Ѱ ө=ө v=v  
 ѱ=ѱ 0x00479=y 0x0047D=0x00461 0x0047F=0x00461 0x0A641=з  
 0x0A643=s 0x0A64B=y 0x0A64D=0x00461 ы=ы ѧ=ѧ 0x10001=y

**language** cz  
**parent** default  
**method** mm,mc,uc  
**replacements** ch=0x10001 Ch=0x10001 CH=0x10001  
**order** а á б с ċ d' é é ě f g h 0x10001 i í j k l m n ň ó p q r ř s š t ť u ú û v  
 w x y ý z ž

<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z á=a é=e í=i ó=o ú=u ý=y č=č d=d ě=e ň=n ř=ř š=š t=t ů=u ž=ž 0x10001=ch
<b>language</b>	da
<b>parent</b>	no
<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z æ ø å
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z å=å æ=æ ø=ø
<b>language</b>	de
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	ä=ae Ä=Ae ö=oe Ö=Oe ü=ue Ü=Ue ß=s
<b>order</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z
<b>language</b>	de-AT
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	a ä b c d e f g h i j k l m n o ö p q r s t u ü v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z ä=ä ö=ö ü=ü
<b>language</b>	de-CH
<b>parent</b>	de
<b>method</b>	mm,mc,uc
<b>replacements</b>	ä=ae Ä=Ae ö=oe Ö=Oe ü=ue Ü=Ue ß=s
<b>order</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z
<b>language</b>	de-DE
<b>parent</b>	de
<b>method</b>	mm,mc,uc
<b>replacements</b>	ä=ae Ä=Ae ö=oe Ö=Oe ü=ue Ü=Ue ß=s
<b>order</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z
<b>language</b>	default
<b>parent</b>	default

<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z
<b>language</b>	deo
<b>parent</b>	de
<b>method</b>	mm,mc,uc
<b>replacements</b>	ä=ae Ä=Ae ö=oe Ö=Oe ü=ue Ü=Ue ß=s
<b>order</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z
<b>language</b>	en
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z
<b>language</b>	es
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	a á b c d e é f g h i í j k l m n ñ o ó p q r s t u ú ü v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z á=a é=e í=i ñ=ñ ó=o ú=u ü=u
<b>language</b>	et
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	a b d e f g h i j k l m n o p r s š z ž t u v w õ ä ö ü x y
<b>entries</b>	a=a b=b d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p r=r s=s t=t u=u v=v w=w x=x y=y z=z ä=ä ö=ö õ=õ ü=ü š=š ž=ž
<b>language</b>	fi
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	a b c d e f g h i j k l m n o p q r s t u v w x y z å ä ö
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z ä=ä å=å ö=ö

[illegible]

<b>order</b>	a á b c 0x10001 d 0x10002 0x10003 e é f g 0x10004 h i í j k l 0x10005 m n 0x10006 o ó ö õ p q r s 0x10007 t 0x10008 u ú ü ũ v w x y z 0x10009
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z á=a é=e í=i ó=o ö=ö ú=u ü=ü õ=õ ũ=ũ 0x10001=cs 0x10002=dz 0x10003=dzs 0x10004=gy 0x10005=ly 0x10006=ny 0x10007=sz 0x10008=ty 0x10009=zs
<b>language</b>	is
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	a á b d ð e é f g h i í j k l m n o ó p r s t u ú v x y ý þ æ ö
<b>entries</b>	a=a b=b d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p r=r s=s t=t u=u v=v x=x y=y á=a æ=æ é=e í=i ð=ð ó=o ö=ö ú=u ý=y þ=þ
<b>language</b>	it
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	a á b c d e é f g h i ì j k l m n o ó ò p q r s t u ú ù v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=i k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=u w=w x=x y=y z=z á=a è=e é=e ì=i í=i ò=o ó=o ù=u ú=u
<b>language</b>	jp
<b>parent</b>	default
<b>method</b>	zm
<b>replacements</b>	0x03041=0x03042 0x03043=0x03044 0x03045=0x03046 0x03047=0x03048 0x03049=0x0304A 0x03063=0x03064 0x03083=0x03084 0x03085=0x03086 0x03087=0x03088
<b>order</b>	0x03042 0x03044 0x03046 0x03048 0x0304A 0x0304B 0x0304D 0x0304F 0x03051 0x03053 0x03055 0x03057 0x03059 0x0305B 0x0305D 0x0305F 0x03061 0x03064 0x03066 0x03068 0x0306A 0x0306B 0x0306C 0x0306D 0x0306E 0x0306F 0x03072 0x03075 0x03078 0x0307B 0x0307E 0x0307F 0x03080 0x03081 0x03082 0x03084 0x03086 0x03088 0x03089 0x0308A 0x0308B 0x0308C 0x0308D 0x0308F 0x03090 0x03091 0x03092 0x03093
<b>entries</b>	0x03042=0x03042 0x03044=0x03044 0x03046=0x03046 0x03048=0x03048 0x0304A=0x0304A 0x0304B=0x0304B 0x0304D=0x0304D 0x0304F=0x0304F 0x03051=0x03051 0x03053=0x03053 0x03055=0x03055 0x03057=0x03057 0x03059=0x03059 0x0305B=0x0305B 0x0305D=0x0305D 0x0305F=0x0305F 0x03061=0x03061 0x03064=0x03064 0x03066=0x03066 0x03068=0x03068

0x0306A=0x0306A 0x0306B=0x0306B 0x0306C=0x0306C 0x0306D=0x0306D  
 0x0306E=0x0306E 0x0306F=0x0306F 0x03072=0x03072 0x03075=0x03075  
 0x03078=0x03078 0x0307B=0x0307B 0x0307E=0x0307E 0x0307F=0x0307F  
 0x03080=0x03080 0x03081=0x03081 0x03082=0x03082 0x03084=0x03084  
 0x03086=0x03086 0x03088=0x03088 0x03089=0x03089 0x0308A=0x0308A  
 0x0308B=0x0308B 0x0308C=0x0308C 0x0308D=0x0308D 0x0308F=0x0308F  
 0x03090=0x03090 0x03091=0x03091 0x03092=0x03092 0x03093=0x03093

**language** kr  
**parent** default  
**method** mm,mc,uc  
**replacements** none  
**order** 0x03131 0x03134 0x03137 0x03139 0x03141 0x03142 0x03145  
 0x03147 0x03148 0x0314A 0x0314B 0x0314C 0x0314D 0x0314E a b c  
 d e f g h i j k l m n o p q r s t u v w x y z  
**entries** a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o  
 p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z

**language** la  
**parent** default  
**method** mm,mc,uc  
**replacements** æ=ae Æ=AE  
**order** a ā ä b c d e ē ě f g h i ĩ j k l m n o ō ö p q r s t u ũ ů v w x y ŷ ŷ z  
**entries** a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o  
 p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y ŷ=y z=z ā=a ä=a ē=e  
 ě=e ĩ=i ĭ=i ō=o ō=o ũ=u ů=u ŷ=y

**language** lt  
**parent** default  
**method** mm,mc,uc  
**replacements** ch=0x10001 CH=0x1000B  
**order** a ą b c 0x10001 č d e ě f g h i ĳ j k l m n o p r s š t u ų ū v z ž  
**entries** a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o  
 p=p r=r s=s t=t u=u v=v y=i z=z ą=a č=č ě=e ĳ=ĳ ų=ų ū=u ų=u  
 ž=ž 0x10001=c

**language** lv  
**parent** default  
**method** mm,mc,uc  
**replacements** none  
**order** a ā b c č d e ē f g ģ h i ĳ j k ķ l ļ m n ņ o ō p r ŀ s š t u ū v z ž  
**entries** a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o  
 p=p r=r s=s t=t u=u v=v z=z ā=a č=č ē=e ģ=ģ ĳ=ķ ķ=ķ ļ=ļ ņ=ņ ō=o  
 ŀ=ŀ š=š ū=u ž=ž

**language** nl  
**parent** default  
**method** mm,mc,uc  
**replacements** ij=y IJ=Y  
**order** a b c d e f g h i j k l m n o p q r s t u v w x y z  
**entries** a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o  
p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z

**language** no  
**parent** default  
**method** mm,mc,uc  
**replacements** none  
**order** a b c d e f g h i j k l m n o p q r s t u v w x y z æ ø å  
**entries** a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o  
p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z å=å æ=æ ø=ø

**language** ocs-scñ  
**parent** default  
**method** mm,mc,uc  
**replacements** ou=0x10001 OU=0x10015 g'=0x10002 G'=0x10016 št=0x10003  
ŠT=0x10017 ju=0x10004 JU=0x10018 ja=0x10005 JA=0x10019  
je=0x10006 JE=0x1001A ję=0x10007 JĘ=0x1001B jq=0x10008  
JQ=0x1001C ks=0x10009 KS=0x1001D ps=0x1000A PS=0x1001E  
th=0x1000B TH=0x1001F šč=0x1000C ŠČ=0x10020  
**order** a b v g d e ž ž z i ĭ 0x10002 k l m n o p r s t u f x o c č š 0x10003  
0x1000C ъ y 0x10001 ъ ě 0x10004 0x10005 0x10006 ę 0x10007 q  
0x10008 0x10009 0x1000A 0x1000B ü  
**entries** a=a b=b c=c d=d e=e f=f g=g i=i k=k l=l m=m n=n o=o p=p r=r  
s=s t=t u=u v=v x=x y=y z=z ĭ=ĭ ü=ü č=č ę=ę ě=ě š=š ž=ž q=q ž=ž  
ъ=ъ ъ=ъ 0x10001=y 0x10002=g' 0x10003=št 0x10004=ju 0x10005=ja  
0x10006=je 0x10007=ję 0x10008=jq 0x10009=ks 0x1000A=ps  
0x1000B=th 0x1000C=šč

**language** pl  
**parent** default  
**method** mm,mc,uc  
**replacements** none  
**order** a ą b c ć d e ę f g h i j k l ł m n ń o ó p q r s ś t u v w x y z ź ż  
**entries** a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o  
p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z ó=ó ą=ą ć=ć ę=ę  
ł=ł ń=ń ś=ś ź=ź ż=ż

**language** pt  
**parent** default  
**method** mm,mc,uc  
**replacements** none



<b>order</b>	a á â ã à b c ç d e é ê f g h i í j k l m n o ó ô õ p q r s t u ú ü v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z à=a á=a â=a ã=a ç=c é=e ê=e í=i ó=o ô=o õ=o ú=u ü=u
<b>language</b>	ro
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	a ä å b c d e f g h i î j k l m n o p q r s ş t ţ u v w x y z
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z â=â î=î ä=ä ş=ş ţ=ţ
<b>language</b>	ru
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	none
<b>order</b>	а б в г д е ё ж з и й к л м н о п р с т у ф х ц ч ш щ ъ ы ь ё э ю я ө в
<b>entries</b>	a=a б=б в=в г=г д=д е=e ж=ж з=з и=и й=й к=к л=л м=м н=н о=o п=п р=p с=c т=t у=y ф=ф х=x ц=ц ч=ч ш=ш щ=щ ъ=ъ ы=ы ь=ь э=э ю=ю я=я ё=e и=и ё=ё ө=ө в=v
<b>language</b>	ru-iso9
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	"=0x10001
<b>order</b>	a b v g d e ё ž z i ð j k l m n o p r s t u f h c č š š " 0x10001 y ' ' ë è û â û â
<b>entries</b>	'=' a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p r=r s=s t=t u=u v=v y=y z=z â=â è=è ë=ë ì=ì û=û č=č ě=ě š=š š=š ž=ž ' ' " " 0x10001="
<b>language</b>	sk
<b>parent</b>	default
<b>method</b>	mm,mc,uc
<b>replacements</b>	dz=0x10001 dz=0x1000B dž=0x10002 dž=0x1000C ch=0x10003 ch=0x1000D
<b>order</b>	a á â b c č d d' 0x10001 0x10002 e é f g h 0x10003 i í j k l l' m n ň o ó ô p q r r' s š t t' u ú v w x y ý z ž
<b>entries</b>	a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z á=a â=a é=e í=i ó=o ô=o ú=u ý=y č=č d'=d l'=l l'=l ň=n r=r š=š t'=t ž=ž 0x10001=dz 0x10002=dž 0x10003=ch

**language** sl  
**parent** default  
**method** mm,mc,uc  
**replacements** none  
**order** a b c č d đ e f g h i j k l m n o p q r s š t u v w x y z ž  
**entries** a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o  
p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z ć=ć č=č đ=đ š=š  
ž=ž

**language** sr  
**parent** default  
**method** mm,mc,uc  
**replacements** none  
**order** a б в г д ђ е ж з и ј к л љ м н њ о п р с т ћ у ф х ц ч ш  
**entries** a=a б=б в=в г=г д=д е=е ж=ж з=з и=и к=к л=л м=м н=н о=o  
п=п р=р с=с т=т у=у ф=ф х=х ц=ц ч=ч ш=ш ћ=ћ ј=ј љ=љ њ=њ  
ћ=ћ ц=ц

**language** sv  
**parent** default  
**method** mm,mc,uc  
**replacements** none  
**order** a b c d e f g h i j k l m n o p q r s t u v w x y z å ä ö  
**entries** a=a b=b c=c d=d e=e f=f g=g h=h i=i j=j k=k l=l m=m n=n o=o  
p=p q=q r=r s=s t=t u=u v=v w=w x=x y=y z=z ä=ä å=å ö=ö

**language** uk  
**parent** default  
**method** mm,mc,uc  
**replacements** none  
**order** а б в г г д е є ж з и і ї к л м н о п р с т у ф х ц ч ш щ ь ю я  
**entries** а=а б=б в=в г=г д=д е=е ж=ж з=з и=и ї=ї к=к л=л м=м н=н  
о=о п=п р=р с=с т=т у=у ф=ф х=х ц=ц ч=ч ш=ш щ=щ ь=ь ю=ю  
я=я є=є і=і ї=ї г=г

## A.4 Verbose counters

```
\usemodule[s-languages-counters]
```

```
\showverbosecounters[language={en,es}]
```

en	es	number
zero		0

one	uno	1
two	dos	2
three	tres	3
four	cuatro	4
five	cinco	5
six	seis	6
seven	siete	7
eight	ocho	8
nine	nueve	9
ten	diez	10
eleven	once	11
twelve	doce	12
thirteen	trece	13
fourteen	catorce	14
fifteen	quince	15
sixteen	dieciséis	16
seventeen	diecisiete	17
eighteen	dieciocho	18
nineteen	diecinueve	19
twenty	veinte	20
	veintiuno	21
	veintidós	22
	veintitrés	23
	veinticuatro	24
	veinticinco	25
	veintiséis	26
	veintisiete	27
	veintiocho	28
	veintinueve	29
thirty	treinta	30
forty	cuarenta	40
fifty	cincuenta	50
sixty	sesenta	60
seventy	setenta	70
eighty	ochenta	80
ninety	noventa	90
hundred	ciento	100
	doscientos	200
	trescientos	300
	cuatrocientos	400
	quinientos	500
	seiscientos	600
	setecientos	700
	ochocientos	800

	novecientos	900
thousand	mil	1000
million	millón	1000000
billion	mil millones	1000000000
trillion	billón	1000000000000