# low level

# TEX

## characters

# Contents

## 1 Introduction

This explanation is part of the low level manuals because in practice users will not have to deal with these matters in MkIV and even less in LMTX. You can skip to the last section for commands.

## 2 History

If we travel back in time to when TEX was written we end up in eight bit character universe. In fact, the first versions assumed seven bits, but for comfortable use with languages other than English that was not sufficient. Support for eight bits permits the usage of so called code pages as supported by operating systems. Although ascii input became kind of the standard soon afterwards, the engine can be set up for different encodings. This is not only true for TEX, but for many of its companions, like MetaFont and therefore MetaPost.[1]

Core components of a TEX engine are hyphenation of words, applying inter-character kerns and build ligatures. In traditional TEX engines those processes are interwoven into the par builder but in LuaTEX these are separate stages. The original approach is the reason that there is a relation between the input encoding and the font encoding: the character in the input is the slot used in a reference to a glyph. When producing the final result (e.g. pdf) there can also be a mapping to an index in a font resource.

```
input A [tex ->] font slot A [backend ->] glyph index A
```

The mapping that TEX does is normally one-to-one but an input character can undergo some transformation. For instance a character beyond ascii 126 can be made active and expand to some character number that then becomes the font slot. So, it is the

---

[1] This remapping to an internal representation (e.g. ebcdic) is not present in LuaTEX where we assume utf8 to be the input encoding. The MetaPost library that comes with LuaTEX still has that code but in LuaMetaTEX it's gone. There one can set up the machinery to be utf8 aware too.

expansion (or meaning) of a character that end up as numeric reference in the glyph node. Virtual fonts can introduce yet another remapping but that's only visible in the backend.

Actually, in LuaTEX the same happens but in practice there is no need to go active because (at least in ConTEXt) we assume a Unicode path so there the font slot is the Unicode got from the utf8 input.

In the eight bit universe macro packages (have to) provide all kind of means to deal with (in the perspective of English) special characters. For instance, `\"a` would put a diaeresis on top of the a or even better, refer to a character in the encoding that the chosen font provides. Because there are some limitations of what can go in an eight bit font, and because in different countries the used TEX fonts evolved kind of independent, we ended up with quite some different variants of fonts. It was only with the Latin Modern project that this became better. Interesting is that when we consider the fact that such a font has often also hardly used symbols (like registered or copyright) coming up with an encoding vector that covers most (latin based) European languages (scripts) is not impossible[2] Special symbols could simply go into a dedicated font, also because these are always accessed via a macro so who cares about the input. It never happened.

Keep in mind that when utf8 is used with eight bit engines, ConTEXt will convert sequences of characters into a slot in a font (depending on the font encoding used which itself depends on the coverage needed). For this every first (possible) byte of a multibyte utf sequence is an active character, which is no big deal because these are outside the ascii range. Normal ascii characters are single byte utf sequences and fall through without treatment.

Anyway, in ConTEXt MkII we dealt with this by supporting mixed encodings, depending on the (local) language, referencing the relevant font. It permits users to enter the text in their preferred input encoding and also get the words properly hyphenated. But we can leave these MkII details behind.

## 3 The heritage

In MkIV we got rid of input and font encodings, although one can still load files in a specific code page.[3] We also kept the means to enter special characters, if only because

---

[2] And indeed in the Latin Modern project we came up with one but it was already to late for it to become popular.

[3] I'm not sure if users ever depend on an input encoding different from utf8.

text editors seldom support(ed) a wide range of visual editing of those. This is why we still have

```
\"u \^a \v{s} \AE \ij \eacute \oslash
```

and many more. The ones with one character names are rather common in the TEX community but it is definitely a weird mix of symbols. The next two are kind of outdated: in these days you delegate that to the font handler, where turning them into ‘single’ character references depends on what the font offers, how it is set up with respect to (for instance) ligatures, and even might depend on language or script.

The ones with the long names partly are tradition, but as we have a lot of them, in MkII they actually serve a purpose. These verbose names are used in the so called encoding vectors and are part of the utf expansion vectors. They are also used in labels so that we have a good indication if what goes in there: remember that in those times editors often didn't show characters, unless the font for display had them, or the operating system somehow provided them from another font. These verbose names are used for latin, greek and cyrillic and for some other scripts and symbols. They take up quite a bit of hash space and the format file.[4]

## 4 The LMTX approach

In the process of tagging all (public) macros in LMTX (which happened in 2020-2021) I wondered if we should keep these one character macros, the references to special characters and the verbose ones. When asked on the mailing list it became clear that users still expect the short ones to be present, often just because old bibTEX files are used that might need them. However, in MkIV and LMTX we load bibTEX files in a way that turn these special character references into proper utf8 input so it makes a weak argument. Anyway, although they could go, for now we keep them because users expect them. However, in LMTX the implementation is somewhat different now, a bit more efficient in terms of hash and memory, potentially a bit less efficient in runtime, but no one will notice that.

A new command has been introduced, the very short `\chr`.

```
\chr {à} \chr {á} \chr {ä}
\chr {`a} \chr {'a} \chr {"a}
\chr {a acute} \chr {a grave} \chr {a umlaut}
```

---

[4] In MkII we have an abstract front-end with respect to encodings and also an abstract backend with respect to supported drivers but both approaches no longer make sense today.

```
\chr {aacute}  \chr {agrave}  \chr {aumlaut}
```

In the first line the composed character using two characters, a base and a so called mark. Actually, one doesn't have to use `\chr` in that case because ConTEXt does already collapse characters for you. The second line looks like the shortcuts `` \` ``, `\'` and `\"`. The third and fourth lines could eventually replace the more symbolic long names, if we feel the need. Watch out: in Unicode input the marks come *after*.

à á ä
à á ä
á à a˘mła˘t
á à a˘mła˘t

Currently the repertoire is somewhat limited but it can be easily be extended. It all depends on user needs (doing Greek and Cyrillic for instance). The reason why we actually save code deep down is that the helpers for this have always been there.[5]

The `\"` commands are now just aliases to more verbose and less hackery looking macros:

| | | | |
|---|---|---|---|
| `\withgrave` | à | `` \` `` | à |
| `\withacute` | á | `\'` | á |
| `\withcircumflex` | â | `\^` | â |
| `\withtilde` | ã | `\~` | ã |
| `\withmacron` | ā | `\=` | ā |
| `\withbreve` | ĕ | `\u` | ĕ |
| `\withdotaccent` | ċ | `\.` | .c |
| `\withdiaeresis` | ë | `\"` | ë |
| `\withring` | ů | `\r` | ů |
| `\withhungarumlaut` | ű | `\H` | ű |
| `\withcaron` | ě | `\v` | ě |
| `\withcedilla` | ȩ | `\c` | ȩ |
| `\withogonek` | ę | `\k` | ę |

Not all fonts have these special characters. Most natural is to have them available as precomposed single glyphs, but it can be that they are just two shapes with the marks anchored to the base. It can even be that the font somehow overlays them, assuming (roughly) equal widths. The `compose` font feature in ConTEXt normally can handle most well.

---

[5] So if needed I can port this approach back to MkIV, but for now we keep it as is because we then have a reference.

An occasional ugly rendering doesn't matter that much: better have something than nothing. But when it's the main language (script) that needs them you'd better look for a font that handles them. When in doubt, in ConTEXt you can enable checking:

| **command** | **equivalent to** |
|---|---|
| `\checkmissingcharacters` | `\enabletrackers[fonts.missing]` |
| `\removemissingcharacters` | `\enabletrackers[fonts.missing=remove]` |
| `\replacemissingcharacters` | `\enabletrackers[fonts.missing=replace]` |
| `\handlemissingcharacters` | `\enabletrackers[fonts.missing={decompose,replace}]` |

The decompose variant will try to turn a composed character into its components so that at least you get something. If that fails it will inject a replacement symbol that stands out so that you can check it. The console also mentions missing glyphs. You don't need to enable this by default[6] but you might occasionally do it when you use a font for the first time.

In LMTX this mechanism has been upgraded so that replacements follow the shape and are actually real characters. The decomposition has not yet been ported back to MkIV.

The full list of commands can be queried when a tracing module is loaded:

```
\usemodule[s][characters-combinations]

\showcharactercombinations
```

We get this list:

| | | | |
|---|---|---|---|
| acute | U+00301 | ´ | `\withacute` |
| breve | U+00306 | ˘ | `\withbreve` |
| caron | U+0030C | ˇ | `\withcaron` |
| caron below | U+0032C | ˬ | `\withcaronbelow` |
| cedilla | U+00327 | ¸ | `\withcedilla` |
| circumflex | U+00302 | ˆ | `\withcircumflex` |
| circumflex below | U+0032D | ꞈ | `\withcircumflexbelow` |
| comma below | U+00327 | ¸ | `\withcommabelow` |
| diaeresis | U+00308 | ¨ | `\withdiaeresis` |
| dieresis | U+00308 | ¨ | `\withdieresis` |
| dot | U+00307 | ˙ | `\withdot` |
| dot below | U+00323 | . | `\withdotbelow` |
| double acute | U+0030B | ˝ | `\withdoubleacute` |

---

[6] There is some overhead involved here.

| | | | |
|---|---|---|---|
| double grave | U+0030F | ̏ | \withdoublegrave |
| double vertical line | U+0030E | ̎ | \withdoubleverticalline |
| grave | U+00300 | ̀ | \withgrave |
| hook | U+00309 | ̉ | \withhook |
| hook below | U+1FA9D | | \withhookbelow |
| hungarumlaut | U+0030B | ̋ | \withhungarumlaut |
| inverted breve | U+00311 | ̑ | \withinvertedbreve |
| line | U+00304 | ̄ | \withline |
| line below | U+00331 | ̱ | \withlinebelow |
| macron | U+00304 | ̄ | \withmacron |
| macron below | U+00331 | ̱ | \withmacronbelow |
| middle dot | U+000B7 | · | \withmiddledot |
| ogonek | U+00328 | ̨ | \withogonek |
| overline | U+00305 | ̅ | |
| ring | U+0030A | ̊ | \withring |
| ring below | U+00325 | ̥ | \withringbelow |
| slash | U+0002F | / | \withslash |
| stroke | U+0002F | / | \withstroke |
| tilde | U+00303 | ̃ | \withtilde |
| tilde below | U+00330 | ̰ | \withtildebelow |
| vertical line | U+0030D | ̍ | \withverticalline |

Some combinations are special for ConTEXt because Unicode doesn't specify decomposition for all composed characters.

## 5 spaces

The engine has no real concept of a space. When the input has one it is turned into a glue, likely with some stretch and shrink. When `\nospaces` is set to one, no glue will be inserted. A value of two will inject a zero width glue. When set to three a glyph will be inserted with the character code set by `\spacechar`.

```
\nospaces\plusthree
\spacechar\underscoreasciicode
\hccode\underscoreasciicode\underscoreasciicode
Where are the spaces?
```

The `hccode` tells the machinery that the underscore is a valid word separator (think compound words).

Where_are_the_spaces?_

# 5 Colofon

| | |
|---|---|
| Author | Hans Hagen |
| ConTEXt | 2024.05.17 16:32 |
| LuaMetaTEX | 211.3 |
| Support | www.pragma-ade.com<br>contextgarden.net |