# The l3str-convert package: string encoding conversions

The LaTeX3 Project*

Released 2018-10-17

## 1 Encoding and escaping schemes

Traditionally, string encodings only specify how strings of characters should be stored as bytes. However, the resulting lists of bytes are often to be used in contexts where only a restricted subset of bytes are permitted (*e.g.*, PDF string objects, URLs). Hence, storing a string of characters is done in two steps.

- The code points ("character codes") are expressed as bytes following a given "encoding". This can be UTF-16, ISO 8859-1, *etc.* See Table 1 for a list of supported encodings.[1]
- Bytes are translated to TeX tokens through a given "escaping". Those are defined for the most part by the `pdf` file format. See Table 2 for a list of escaping methods supported.[2]

*E-mail: latex-team@latex-project.org

[1] Encodings and escapings will be added as they are requested.

Table 1: Supported encodings. Non-alphanumeric characters are ignored, and capital letters are lower-cased before searching for the encoding in this list.

| ⟨*Encoding*⟩ | description |
|---|---|
| `utf8` | UTF-8 |
| `utf16` | UTF-16, with byte-order mark |
| `utf16be` | UTF-16, big-endian |
| `utf16le` | UTF-16, little-endian |
| `utf32` | UTF-32, with byte-order mark |
| `utf32be` | UTF-32, big-endian |
| `utf32le` | UTF-32, little-endian |
| `iso88591`, `latin1` | ISO 8859-1 |
| `iso88592`, `latin2` | ISO 8859-2 |
| `iso88593`, `latin3` | ISO 8859-3 |
| `iso88594`, `latin4` | ISO 8859-4 |
| `iso88595` | ISO 8859-5 |
| `iso88596` | ISO 8859-6 |
| `iso88597` | ISO 8859-7 |
| `iso88598` | ISO 8859-8 |
| `iso88599`, `latin5` | ISO 8859-9 |
| `iso885910`, `latin6` | ISO 8859-10 |
| `iso885911` | ISO 8859-11 |
| `iso885913`, `latin7` | ISO 8859-13 |
| `iso885914`, `latin8` | ISO 8859-14 |
| `iso885915`, `latin9` | ISO 8859-15 |
| `iso885916`, `latin10` | ISO 8859-16 |
| `clist` | comma-list of integers |
| ⟨*empty*⟩ | native (Unicode) string |

Table 2: Supported escapings. Non-alphanumeric characters are ignored, and capital letters are lower-cased before searching for the escaping in this list.

| ⟨*Escaping*⟩ | description |
|---|---|
| `bytes`, or empty | arbitrary bytes |
| `hex`, `hexadecimal` | byte = two hexadecimal digits |
| `name` | see `\pdfescapename` |
| `string` | see `\pdfescapestring` |
| `url` | encoding used in URLs |

## 2 Conversion functions

\str_set_convert:Nnnn
\str_gset_convert:Nnnn

\str_set_convert:Nnnn ⟨*str var*⟩ {⟨*string*⟩} {⟨*name 1*⟩} {⟨*name 2*⟩}

This function converts the ⟨*string*⟩ from the encoding given by ⟨*name 1*⟩ to the encoding given by ⟨*name 2*⟩, and stores the result in the ⟨*str var*⟩. Each ⟨*name*⟩ can have the form ⟨*encoding*⟩ or ⟨*encoding*⟩/⟨*escaping*⟩, where the possible values of ⟨*encoding*⟩ and ⟨*escaping*⟩ are given in Tables 1 and 2, respectively. The default escaping is to input and output bytes directly. The special case of an empty ⟨*name*⟩ indicates the use of "native" strings, 8-bit for pdfTEX, and Unicode strings for the other two engines.

For example,

```
\str_set_convert:Nnnn \l_foo_str { Hello! } { } { utf16/hex }
```

results in the variable `\l_foo_str` holding the string `FEFF00480065006C006C006F0021`. This is obtained by converting each character in the (native) string `Hello!` to the UTF-16 encoding, and expressing each byte as a pair of hexadecimal digits. Note the presence of a (big-endian) byte order mark "FEFF, which can be avoided by specifying the encoding `utf16be/hex`.

An error is raised if the ⟨*string*⟩ is not valid according to the ⟨*escaping 1*⟩ and ⟨*encoding 1*⟩, or if it cannot be reencoded in the ⟨*encoding 2*⟩ and ⟨*escaping 2*⟩ (for instance, if a character does not exist in the ⟨*encoding 2*⟩). Erroneous input is replaced by the Unicode replacement character "FFFD, and characters which cannot be reencoded are replaced by either the replacement character "FFFD if it exists in the ⟨*encoding 2*⟩, or an encoding-specific replacement character, or the question mark character.

\str_set_convert:NnnnTF
\str_gset_convert:NnnnTF

\str_set_convert:NnnnTF ⟨*str var*⟩ {⟨*string*⟩} {⟨*name 1*⟩} {⟨*name 2*⟩} {⟨*true code*⟩} {⟨*false code*⟩}

As `\str_set_convert:Nnnn`, converts the ⟨*string*⟩ from the encoding given by ⟨*name 1*⟩ to the encoding given by ⟨*name 2*⟩, and assigns the result to ⟨*str var*⟩. Contrarily to `\str_set_convert:Nnnn`, the conditional variant does not raise errors in case the ⟨*string*⟩ is not valid according to the ⟨*name 1*⟩ encoding, or cannot be expressed in the ⟨*name 2*⟩ encoding. Instead, the ⟨*false code*⟩ is performed.

## 3 Possibilities, and things to do

Encoding/escaping-related tasks.

- In XeTEX/LuaTEX, would it be better to use the `^^^^....` approach to build a string from a given list of character codes? Namely, within a group, assign `0-9a-f` and all characters we want to category "other", then assign `^` the category superscript, and use `\scantokens`.
- Change `\str_set_convert:Nnnn` to expand its last two arguments.
- Describe the internal format in the code comments. Refuse code points in ["D800, "DFFF] in the internal representation?
- Add documentation about each encoding and escaping method, and add examples.
- The `hex` unescaping should raise an error for odd-token count strings.

- Decide what bytes should be escaped in the `url` escaping. Perhaps `!'()*-./0123456789_` are safe, and all other characters should be escaped?
- Automate generation of 8-bit mapping files.
- Change the framework for 8-bit encodings: for decoding from 8-bit to Unicode, use 256 integer registers; for encoding, use a tree-box.
- More encodings (see Heiko's stringenc). CESU?
- More escapings: ASCII85, shell escapes, lua escapes, *etc.*?

# Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.