Glossary of TEX terms used to describe LATEX3 functions

The LATEX Project*

Released 2026-11-10

This file describes aspects of TEX programming that are relevant in a expl3 context.

1 Reading a file

Tokenization.

Treatment of spaces, such as the trap that `\~~a` is equivalent to `\~a` in expl3 syntax, or that `~` fails to give a space at the beginning of a line.

2 Structure of tokens

We refer to the documentation of `l3token` for a complete description of all TEX tokens. We distinguish the meaning of the token, which controls the expansion of the token and its effect on TEX's state, and its shape, which is used when comparing token lists such as for delimited arguments. At any given time two tokens of the same shape automatically have the same meaning, but the converse does not hold, and the meaning associated with a given shape change when doing assignments.

Apart from a few exceptions, a token has one of the following shapes.

- A control sequence, characterized by the sequence of characters that constitute its name: for instance, `\use:n` is a five-letter control sequence.
- An active character token, characterized by its character code (between 0 and 1114111 for LuaTEX and XeTEX and less for other engines) and category code 13.
- A character token such as `A` or `#`, characterized by its character code and category code (one of 1, 2, 3, 4, 6, 7, 8, 10, 11 or 12 whose meaning is described below).

The meaning of a (non-active) character token is fixed by its category code (and character code) and cannot be changed. We call these tokens *explicit* character tokens. Category codes that a character token can have are listed below by giving a sample output of the TEX primitive `\meaning`, together with their expl3 names and most common example:

1 begin-group character (`group_begin`, often `{`),

2 end-group character (`group_end`, often `}`),

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

3 math shift character (`math_toggle`, often `$`),

4 alignment tab character (`alignment`, often `&`),

6 macro parameter character (`parameter`, often `#`),

7 superscript character (`math_superscript`, often `^`),

8 subscript character (`math_subscript`, often `_`),

10 blank space (`space`, often character code 32),

11 the letter (`letter`, such as `A`),

12 the character (`other`, such as `0`).

Category code 13 (`active`) is discussed below. Input characters can also have several other category codes which do not lead to character tokens for later processing: 0 (`escape`), 5 (`end_line`), 9 (`ignore`), 14 (`comment`), and 15 (`invalid`).

The meaning of a control sequence or active character can be identical to that of any character token listed above (with any character code), and we call such tokens *implicit* character tokens. The meaning is otherwise in the following list:

- a macro, used in expl3 for most functions and some variables (`tl`, `fp`, `seq`, ...),
- a primitive such as `\def` or `\topmark`, used in expl3 for some functions,
- a register such as `\count123`, used in expl3 for the implementation of some variables (`int`, `dim`, ...),
- a constant integer such as `\char"56` or `\mathchar"121`, used when defining a constant using `\int_const:Nn`,
- a font selection command,
- undefined.

Macros can be `\protected` or not, `\long` or not (the opposite of what expl3 calls `nopar`), and `\outer` or not (unused in expl3). Their `\meaning` takes the form

⟨*prefix*⟩ `macro:`⟨*argument*⟩`->`⟨*replacement*⟩

where ⟨*prefix*⟩ is among `\protected\long\outer`, ⟨*argument*⟩ describes parameters that the macro expects, such as `#1#2#3`, and ⟨*replacement*⟩ describes how the parameters are manipulated, such as `\int_eval:n{#2+#1*#3}`. This information can be accessed by `\cs_prefix_spec:N`, `\cs_parameter_spec:N`, `\cs_replacement_spec:N`.

When a macro takes an undelimited argument, explicit space characters (with character code 32 and category code 10) are ignored. If the following token is an explicit character token with category code 1 (begin-group) and an arbitrary character code, then TEX scans ahead to obtain an equal number of explicit character tokens with category code 1 (begin-group) and 2 (end-group), and the resulting list of tokens (with outer braces removed) becomes the argument. Otherwise, a single token is taken as the argument for the macro: we call such single tokens "N-type", as they are suitable to be used as an argument for a function with the signature `:N`.

When a macro takes a delimited argument TEX scans ahead until finding the delimiter (outside any pairs of begin-group/end-group explicit characters), and the resulting list of tokens (with outer braces removed) becomes the argument. Note that explicit space characters at the start of the argument are *not* ignored in this case (and they prevent brace-stripping).

3 Handling of hash tokens

TEX uses the hash (octothorpe) character `#` to denote parameters for macros: these must be numbered sequentially. To allow handling of nested macros, TEX requires that for each nesting level, hash tokens are doubled. For example

```
s_new:Npn \mypkg_outer:N #1
{
  \cs_new:Npn \mypkg_inner:N ##1
    {
      #1
      ##1
    }
}
```

would define both `\mypkg_outer:N` and `\mypkg_inner:N` as taking exactly one argument. If we then do

```
ypkg_outer:N \foo
s_show:N \mypkg_inner:N
```

TEX will report

```
\mypkg_inner:N=\long macro:#1->\foo #1.
```

i.e., the hash is not doubled, but is now the parameter of this macro.

Exactly the same concept applies to anywhere that inline code is nested in expl3, for example inline mapping code, key definitions, etc.