The l3tl-build package: building token lists*

The LaTeX3 Project†

Released 2012/04/23

1 l3tl-build documentation

This module provides no user function.

There are two main ways of building token lists from individual tokens. Either in one go within an `x`-expanding assignment, or by repeatedly using `\tl_put_right:Nn`. The first method takes a linear time, but only allows expandable operations. The second method takes a time quadratic in the length of the token list, but allows expandable and non-expandable operations.

The goal of this module is to provide functions to build a token list piece by piece in linear time, while allowing non-expandable operations. This is achieved by abusing `\toks`: adding some tokens to the token list is done by storing them in a free token register (time $O(1)$ for each such operation). Those token registers are only put together at the end, within an `x`-expanding assignment, which takes a linear time.[1] Of course, all this must be done in a group: we can't go and clobber the values of legitimate `\toks` used by LaTeX 2_ε.

Since none of the current applications need the ability to insert material on the left of the token list, I have not implemented that. This could be done for instance by using odd-numbered `\toks` for the left part, and even-numbered `\toks` for the right part.

*This file describes v3570, last revised 2012/04/23.

†E-mail: latex-team@latex-project.org

[1] If we run out of token registers, then the currently filled-up `\toks` are put together in a temporary token list, and cleared, and we ultimately use `\tl_put_right:Nx` to put those chunks together. Hence the true asymptotic is quadratic, with a very small constant.

1.1 Internal functions

\tl_set_build:Nw
\tl_gset_build:Nw
\tl_set_build_x:Nw
\tl_gset_build_x:Nw

```
\tl_set_build:Nw ⟨tl var⟩ ...
\tl_build_one:n {⟨tokens_1⟩} ...
\tl_build_one:n {⟨tokens_2⟩} ...
...
\tl_build_end:
```

Defines the ⟨*tl var*⟩ to contain the contens of ⟨*tokens1*⟩ followed by ⟨*tokens2*⟩, *etc.* This is built in such a way to be more efficient than repeatedly using `\tl_put_right:Nn`. The code in "`...`" does not need to be expandable. The commands `\tl_set_build:Nw` and `\tl_build_end:` start and end a group. The assignment to the ⟨*tl var*⟩ occurs just after the end of that group, using `\tl_set:Nn`, `\tl_gset:Nn`, `\tl_set:Nx`, or `\tl_gset:Nx`.

\tl_build_one:n
\tl_build_one:(o|x)

```
\tl_build_one:n {⟨tokens⟩}
```

This function may only be used within the scope of a `\tl_set_build:Nw` function. It adds the ⟨*tokens*⟩ on the right of the current token list.

\tl_build_end:

Ends the scope started by `\tl_set_build:Nw`, and performs the relevant assignment.

2 l3tl-build implementation

```
⟨*initex | package⟩
\ProvidesExplPackage
  {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}
```

2.1 Variables and helper functions

\l_tl_build_start_index_int
\l_tl_build_index_int

Integers pointing to the starting index (currently always starts at zero), and the current index. The corresponding `\toks` are accessed directly by number.

```
\int_new:N \l_tl_build_start_index_int
\int_new:N \l_tl_build_index_int
```

(*End definition for* `\l_tl_build_start_index_int` *and* `\l_tl_build_index_int`*. These variables are documented on page* **??**.)

\l_tl_build_result_tl

The resulting token list is normally built in one go by unpacking all `\toks` in some range. In the rare cases where there are too many `\tl_build_one:n` commands, leading to the depletion of registers, the contents of the current set of `\toks` is unpacked into `\l_tl_build_result_tl`. This prevents overflow from affecting the end-user (beyond an obvious performance hit).

```
\tl_new:N \l_tl_build_result_tl
```

(*End definition for* `\l_tl_build_result_tl`*. This variable is documented on page* **??**.)

\tl_build_unpack:
\tl_build_unpack_loop:w

The various pieces of the token list are built in \toks from the start_index (inclusive) to the (current) index (excluded). Those \toks are unpacked and stored in order in the result token list. Optimizations would be possible here, for instance, unpacking 10 \toks at a time with a macro expanding to \the\toks#10...\the\toks#19, but this should be kept for much later.

```
\cs_new_protected_nopar:Npn \tl_build_unpack:
  {
    \tl_put_right:Nx \l_tl_build_result_tl
      {
        \exp_after:wN \tl_build_unpack_loop:w
          \int_use:N \l_tl_build_start_index_int ;
        \prg_break_point:n { }
      }
  }
\cs_new:Npn \tl_build_unpack_loop:w #1 ;
  {
    \if_num:w #1 = \l_tl_build_index_int
      \exp_after:wN \prg_map_break:
    \fi:
    \tex_the:D \tex_toks:D #1 \exp_stop_f:
    \exp_after:wN \tl_build_unpack_loop:w
      \int_use:N \int_eval:w #1 + \c_one ;
  }
```

(*End definition for* \tl_build_unpack:. *This function is documented on page* ??.)

2.2 Building the token list

\tl_set_build:Nw
\tl_set_build_x:Nw
\tl_gset_build:Nw
\tl_gset_build_x:Nw
\tl_set_build_aux:NNw

Similar to what is done for coffins: redefine some command, here \tl_build_end_aux:n to hold the relevant assignment (see \tl_build_end: for details). Then initialize the start index and the current index at zero, and empty the result token list.

```
\cs_new_protected_nopar:Npn \tl_set_build:Nw
  { \tl_set_build_aux:NNw \tl_set:Nn }
\cs_new_protected_nopar:Npn \tl_set_build_x:Nw
  { \tl_set_build_aux:NNw \tl_set:Nx }
\cs_new_protected_nopar:Npn \tl_gset_build:Nw
  { \tl_set_build_aux:NNw \tl_gset:Nn }
\cs_new_protected_nopar:Npn \tl_gset_build_x:Nw
  { \tl_set_build_aux:NNw \tl_gset:Nx }
\cs_new_protected:Npn \tl_set_build_aux:NNw #1#2
  {
    \group_begin:
      \cs_set_nopar:Npn \tl_build_end_assignment:n
        { \group_end: #1 #2 }
      \int_zero:N \l_tl_build_start_index_int
      \int_zero:N \l_tl_build_index_int
      \tl_clear:N \l_tl_build_result_tl
  }
```

(*End definition for* \tl_set_build:Nw *and others. These functions are documented on page* 2.)

\tl_build_end:
\tl_build_end_assignment:n

When we are done building a token list, unpack all \toks into the result token list, and expand this list before closing the group. The \tl_build_end_assignment:n function is defined by \tl_set_build_aux:NNw to end the group and hold the relevant assignment. Its value outside is irrelevant, but just in case, we set it to a function which would clean up the contents of \l_tl_build_result_tl.

```
\cs_new_protected_nopar:Npn \tl_build_end:
  {
      \tl_build_unpack:
      \exp_args:No
    \tl_build_end_assignment:n \l_tl_build_result_tl
  }
\cs_new_eq:NN \tl_build_end_assignment:n \use_none:n
```

(*End definition for* \tl_build_end:. *This function is documented on page* ??.)

\tl_build_one:n
\tl_build_one:o
\tl_build_one:x

Store the tokens in a free \toks, then move the pointer to the next one. If we overflow, unpack the current \toks, and reset the current index, preparing to fill more \toks. This could be optimized by avoiding to read #1, using \afterassignment.

```
\cs_new_protected:Npn \tl_build_one:n #1
  {
    \tex_toks:D \l_tl_build_index_int {#1}
    \tex_advance:D \l_tl_build_index_int \c_one
    \if_num:w \l_tl_build_index_int > \c_max_register_int
      \tl_build_unpack:
      \l_tl_build_index_int \l_tl_build_start_index_int
    \fi:
  }
\cs_new_protected:Npn \tl_build_one:o #1
  {
    \tex_toks:D \l_tl_build_index_int \exp_after:wN {#1}
    \tex_advance:D \l_tl_build_index_int \c_one
    \if_num:w \l_tl_build_index_int > \c_max_register_int
      \tl_build_unpack:
      \l_tl_build_index_int \l_tl_build_start_index_int
    \fi:
  }
\cs_new_protected:Npn \tl_build_one:x #1
  { \use:x { \tl_build_one:n {#1} } }
```

(*End definition for* \tl_build_one:n, \tl_build_one:o, *and* \tl_build_one:x. *These functions are documented on page* ??.)

⟨/initex | package⟩

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.

E

F

G

I

L

P

T

U