

The `l3tl-build` package: building token lists*

The L^AT_EX3 Project[†]

Released 2012/02/26

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 L^AT_EX 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 v3471, last revised 2012/02/26.

[†]E-mail: latex-team@latex-project.org

¹If 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 {\tokens1} ...
\tl_build_one:n {\tokens2} ...
...
\tl_build_end:
```

Defines the $\langle tl\ var\rangle$ to contain the contents of $\langle tokens1\rangle$ followed by $\langle tokens2\rangle$, *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 $\langle tl\ var\rangle$ 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 $\langle tokens\rangle$ 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

```
1 <*initex | package>
2 \ProvidesExplPackage
3   {\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.

```
4 \int_new:N \l_tl_build_start_index_int
5 \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).

```
6 \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:` 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.

```

7 \cs_new_protected_nopar:Npn \tl_build_unpack:
8 {
9   \tl_put_right:Nx \l_tl_build_result_tl
10   {
11     \exp_after:wN \tl_build_unpack_loop:w
12     \int_use:N \l_tl_build_start_index_int ;
13     \prg_break_point:n { }
14   }
15 }
16 \cs_new:Npn \tl_build_unpack_loop:w #1 ;
17 {
18   \if_num:w #1 = \l_tl_build_index_int
19   \exp_after:wN \prg_map_break:
20   \fi:
21   \tex_the:D \tex_toks:D #1 \exp_stop_f:
22   \exp_after:wN \tl_build_unpack_loop:w
23   \int_use:N \int_eval:w #1 + \c_one ;
24 }

```

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

2.2 Building the token list

`\tl_set_build:Nw` 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 `\tl_set_build_x:Nw` start index and the current index at zero, and empty the `result` token list.

```

\tl_gset_build:Nw
\tl_gset_build_x:Nw
\tl_set_build_aux:NNw
25 \cs_new_protected_nopar:Npn \tl_set_build:Nw
26 { \tl_set_build_aux:NNw \tl_set:Nn }
27 \cs_new_protected_nopar:Npn \tl_set_build_x:Nw
28 { \tl_set_build_aux:NNw \tl_set:Nx }
29 \cs_new_protected_nopar:Npn \tl_gset_build:Nw
30 { \tl_set_build_aux:NNw \tl_gset:Nn }
31 \cs_new_protected_nopar:Npn \tl_gset_build_x:Nw
32 { \tl_set_build_aux:NNw \tl_gset:Nx }
33 \cs_new_protected:Npn \tl_set_build_aux:NNw #1#2
34 {
35   \group_begin:
36   \cs_set_nopar:Npn \tl_build_end_assignment:n
37   { \group_end: #1 #2 }
38   \int_zero:N \l_tl_build_start_index_int
39   \int_zero:N \l_tl_build_index_int
40   \tl_clear:N \l_tl_build_result_tl
41 }

```

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

<code>\tl_build_end:</code> <code>\tl_build_end_assignment:n</code>	<p>When we are done building a token list, unpack all <code>\toks</code> into the <code>result</code> token list, and expand this list before closing the group. The <code>\tl_build_end_assignment:n</code> function is defined by <code>\tl_set_build_aux:NNw</code> 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 <code>\l_tl_build_result_tl</code>.</p> <pre> 42 \cs_new_protected_nopar:Npn \tl_build_end: 43 { 44 \tl_build_unpack: 45 \exp_args:No 46 \tl_build_end_assignment:n \l_tl_build_result_tl 47 } 48 \cs_new_eq:NN \tl_build_end_assignment:n \use_none:n </pre> <p><i>(End definition for <code>\tl_build_end:</code>. This function is documented on page ??.)</i></p>
<code>\tl_build_one:n</code> <code>\tl_build_one:o</code> <code>\tl_build_one:x</code>	<p>Store the tokens in a free <code>\toks</code>, then move the pointer to the next one. If we overflow, unpack the current <code>\toks</code>, and reset the current index, preparing to fill more <code>\toks</code>. This could be optimized by avoiding to read <code>#1</code>, using <code>\afterassignment</code>.</p> <pre> 49 \cs_new_protected:Npn \tl_build_one:n #1 50 { 51 \tex_toks:D \l_tl_build_index_int {#1} 52 \tex_advance:D \l_tl_build_index_int \c_one 53 \if_num:w \l_tl_build_index_int > \c_max_register_int 54 \tl_build_unpack: 55 \l_tl_build_index_int \l_tl_build_start_index_int 56 \fi: 57 } 58 \cs_new_protected:Npn \tl_build_one:o #1 59 { 60 \tex_toks:D \l_tl_build_index_int \exp_after:wN {#1} 61 \tex_advance:D \l_tl_build_index_int \c_one 62 \if_num:w \l_tl_build_index_int > \c_max_register_int 63 \tl_build_unpack: 64 \l_tl_build_index_int \l_tl_build_start_index_int 65 \fi: 66 } 67 \cs_new_protected:Npn \tl_build_one:x #1 68 { \use:x { \tl_build_one:n {#1} } } </pre> <p><i>(End definition for <code>\tl_build_one:n</code>, <code>\tl_build_one:o</code>, and <code>\tl_build_one:x</code>. These functions are documented on page ??.)</i></p> <pre> 69 </initex package> </pre>

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.

C

`\c_max_register_int` 53, 62₄

