

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

The L<sup>A</sup>T<sub>E</sub>X3 Project<sup>†</sup>

Released 2011/12/08

## 1 `l3tl-build` documentation

This module provides no user function: it is meant for kernel use only.

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.<sup>1</sup> Of course, all this must be done in a group: we can't go and clobber the values of legitimate `\toks` used by L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>.

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 v3039, last revised 2011/12/08.

<sup>†</sup>E-mail: [latex-team@latex-project.org](mailto:latex-team@latex-project.org)

<sup>1</sup>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

---

<code>\__tl_build:Nw</code>	<code>\__tl_build:Nw &lt;tl var&gt; ...</code>
<code>\__tl_gbuild:Nw</code>	<code>\__tl_build_one:n {\tokens<sub>1</sub>} ...</code>
<code>\__tl_build_x:Nw</code>	<code>\__tl_build_one:n {\tokens<sub>2</sub>} ...</code>
<code>\__tl_gbuild_x:Nw</code>	<code>...</code>

---

Defines the `<tl var>` to contain the contents 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_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`.

---

<code>\__tl_build_one:n</code>	<code>\__tl_build_one:n {\tokens}</code>
<code>\__tl_build_one:(o x)</code>	

---

This function may only be used within the scope of a `\__tl_build:Nw` function. It adds the `<tokens>` on the right of the current token list.

---

<code>\__tl_build_end:</code>	Ends the scope started by <code>\__tl_build:Nw</code> , and performs the relevant assignment.
-------------------------------	---

---

## 2 l3tl-build implementation

```

1 <*initex | package>
2 <@@=tlbuild>
3 \ProvidesExplPackage
4   {\ExplFileName}{\ExplFileDate}{\ExplFileVersion}{\ExplFileDescription}

```

### 2.1 Variables and helper functions

<code>\l__tl_build_start_index_int</code>	Integers pointing to the starting index (currently always starts at zero), and the current
<code>\l__tl_build_index_int</code>	index. The corresponding <code>\toks</code> are accessed directly by number.

```

5 \int_new:N \l__tl_build_start_index_int
6 \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 ??.)

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

```

7 \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.

```

8 \cs_new_protected_nopar:Npn \__tl_build_unpack:
9 {
10   \tl_put_right:Nx \l__tl_build_result_tl
11   {
12     \exp_after:wN \__tl_build_unpack_loop:w
13     \int_use:N \l__tl_build_start_index_int ;
14     \__prg_break_point:
15   }
16 }
17 \cs_new:Npn \__tl_build_unpack_loop:w #1 ;
18 {
19   \if_int_compare:w #1 = \l__tl_build_index_int
20   \exp_after:wN \__prg_break:
21   \fi:
22   \tex_the:D \tex_toks:D #1 \exp_stop_f:
23   \exp_after:wN \__tl_build_unpack_loop:w
24   \int_use:N \__int_eval:w #1 + \c_one ;
25 }

```

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

## 2.2 Building the token list

`__tl_build:Nw` Similar to what is done for coffins: redefine some command, here `\__tl_build_end_`  
`__tl_build_x:Nw` `aux:n` to hold the relevant assignment (see `\__tl_build_end:` for details). Then initial-  
`__tl_gbuild:Nw` ize the start index and the current index at zero, and empty the `result` token list.  
`__tl_gbuild_x:Nw`

```

26 \cs_new_protected_nopar:Npn \__tl_build:Nw
27 { \__tl_build_aux:NNw \tl_set:Nn }
28 \cs_new_protected_nopar:Npn \__tl_build_x:Nw
29 { \__tl_build_aux:NNw \tl_set:Nx }
30 \cs_new_protected_nopar:Npn \__tl_gbuild:Nw
31 { \__tl_build_aux:NNw \tl_gset:Nn }
32 \cs_new_protected_nopar:Npn \__tl_gbuild_x:Nw
33 { \__tl_build_aux:NNw \tl_gset:Nx }
34 \cs_new_protected:Npn \__tl_build_aux:NNw #1#2
35 {
36   \group_begin:
37   \cs_set_nopar:Npn \__tl_build_end_assignment:n
38   { \group_end: #1 #2 }
39   \int_zero:N \l__tl_build_start_index_int
40   \int_zero:N \l__tl_build_index_int
41   \tl_clear:N \l__tl_build_result_tl
42 }

```

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

`\_tl\_build\_end:` 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\_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`.

`\_tl\_build\_end\_assignment:n`

```

43 \cs_new_protected_nopar:Npn \_tl\_build\_end:
44 {
45   \_tl\_build\_unpack:
46   \exp\_args:No
47   \_tl\_build\_end\_assignment:n \l\_tl\_build\_result\_tl
48 }
49 \cs_new_eq:NN \_tl\_build\_end\_assignment:n \use\_none:n
(End definition for \_tl\_build\_end:. This function is documented on page 2.)

```

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

`\_tl\_build\_one:o`  
`\_tl\_build\_one:x`

```

50 \cs_new_protected:Npn \_tl\_build\_one:n #1
51 {
52   \tex\_toks:D \l\_tl\_build\_index\_int {#1}
53   \tex\_advance:D \l\_tl\_build\_index\_int \c\_one
54   \if\_int\_compare:w \l\_tl\_build\_index\_int > \c\_max\_register\_int
55     \_tl\_build\_unpack:
56     \l\_tl\_build\_index\_int \l\_tl\_build\_start\_index\_int
57   \fi:
58 }
59 \cs_new_protected:Npn \_tl\_build\_one:o #1
60 {
61   \tex\_toks:D \l\_tl\_build\_index\_int \exp\_after:wN {#1}
62   \tex\_advance:D \l\_tl\_build\_index\_int \c\_one
63   \if\_int\_compare:w \l\_tl\_build\_index\_int > \c\_max\_register\_int
64     \_tl\_build\_unpack:
65     \l\_tl\_build\_index\_int \l\_tl\_build\_start\_index\_int
66   \fi:
67 }
68 \cs_new_protected:Npn \_tl\_build\_one:x #1
69 { \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 ??.)
70 </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.

### Symbols

`\_int\_eval:w` ..... 24<sub>4</sub>

<code>\__prg_break:</code> .....	20		
<code>\__prg_break_point:</code> .....	14		
<code>\__tl_build:Nw</code> .....	2, 26, 26		
<code>\__tl_build_aux:NNw</code> .	26, 27, 29, 31, 33, 34		
<code>\__tl_build_end:</code> .....	2, 43, 43		
<code>\__tl_build_end_assignment:n</code> .....	37, 43, 47, 49		
<code>\__tl_build_one:n</code> .....	2, 50, 50, 69		
<code>\__tl_build_one:o</code> .....	50, 59		
<code>\__tl_build_one:x</code> .....	50, 68		
<code>\__tl_build_unpack:</code> .....	8, 8, 45, 55, 64		
<code>\__tl_build_unpack_loop:w</code> ..	8, 12, 17, 23		
<code>\__tl_build_x:Nw</code> .....	2, 26, 28		
<code>\__tl_gbuild:Nw</code> .....	2, 26, 30		
<code>\__tl_gbuild_x:Nw</code> .....	2, 26, 32		
		<b>G</b>	
		<code>\group_begin:</code> .....	36
		<code>\group_end:</code> .....	38
		<b>I</b>	
		<code>\if_int_compare:w</code> .....	19, 54, 63
		<code>\int_new:N</code> .....	5, 6
		<code>\int_use:N</code> .....	13, 24
		<code>\int_zero:N</code> .....	39, 40
		<b>L</b>	
		<code>\l__tl_build_index_int</code> .....	5, 6, 19, 40, 52, 53, 54, 56, 61, 62, 63, 65
		<code>\l__tl_build_result_tl</code> ..	7, 7, 10, 41, 47
		<code>\l__tl_build_start_index_int</code> .....	5, 5, 13, 39, 56, 65
		<b>C</b>	
<code>\c_max_register_int</code> .....	54, 63		
<code>\c_one</code> .....	24, 53, 62		
<code>\cs_new:Npn</code> .....	17		
<code>\cs_new_eq:NN</code> .....	49		
<code>\cs_new_protected:Npn</code> ....	34, 50, 59, 68		
<code>\cs_new_protected_nopar:Npn</code> .....	8, 26, 28, 30, 32, 43		
<code>\cs_set_nopar:Npn</code> .....	37		
		<b>P</b>	
		<code>\ProvidesExplPackage</code> .....	3
		<b>T</b>	
		<code>\tex_advance:D</code> .....	53, 62
		<code>\tex_the:D</code> .....	22
		<code>\tex_toks:D</code> .....	22, 52, 61
		<code>\tl_clear:N</code> .....	41
		<code>\tl_gset:Nn</code> .....	31
		<code>\tl_gset:Nx</code> .....	33
		<code>\tl_new:N</code> .....	7
		<code>\tl_put_right:Nx</code> .....	10
		<code>\tl_set:Nn</code> .....	27
		<code>\tl_set:Nx</code> .....	29
		<b>U</b>	
		<code>\use:x</code> .....	69
		<code>\use_none:n</code> .....	49
		<b>F</b>	
<code>\fi:</code> .....	21, 57, 66		
		<b>E</b>	
<code>\exp_after:wN</code> .....	12, 20, 23, 61		
<code>\exp_args:No</code> .....	46		
<code>\exp_stop_f:</code> .....	22		
<code>\ExplFileDate</code> .....	4		
<code>\ExplFileDescription</code> .....	4		
<code>\ExplFileName</code> .....	4		
<code>\ExplFileVersion</code> .....	4		