The xstacks Package
Version 1.1

Alceu Frigeri[*]

February 2026

Abstract

Originally this package aimed at solving the problem of preserving some tokens beyond a variable number of nested groups. Though, after some further testing and optimization, it became clear that (a) there was a lot of room for improvement (stacks) and (b) there was a better way to implement the aftergroup mechanism. As such, there are two new packages at CTAN, *`tokgroupmark`*[3] which implements (b) and *`tokglobalstack`*[2] which implements (a).

This package now loads these two, and set a series of aliases such that the "user visible commands" remain the same (but the user is invited to use one of the other two, instead of this). Moreover, the benchmark part got updated (see 7).

Contents

1 Introduction

(Original rationality behind this package) Sometimes one needs to preserve the value of some variables beyond a local group. In simple cases it's enough to use `\group_insert_after:N` or `\aftergroup`, if you know how many nested groups you are in.

But, sometimes you don't have this information (see [1], for instance) in which case you have a few options:

- use global variables, or
- implement an after group strategy, as suggested by Carlisle, or
- use a (global) stack.

The global variables way is, of course, the fastest (and easiest) if you don't have to worry about reentrant coding (like when you have nested groups inside an environment, which might get nested into itself).

---

[*] `https://github.com/alceu-frigeri/xstacks`

2 Simple mark point after group variant

`\xstacks_groupmark:`
`\xstacks_aftergroup:N`

`\xstacks_groupmark:`
`\xstacks_aftergroup:N {⟨token⟩}`

These will use a single, internal, variable to 'track' the target group level. Better said, upon calling `\xstacks_groupmark:` the current group will be saved (local assignment), so that, later on, `\xstacks_aftergroup:N` can be called from nested groups and ⟨`token`⟩ will be pushed into the marked group.

> ***Note:*** Since all assignments are local, it's possible to have multiple marks, for instance, one at group level 2, another at group level 5, so that anything saved with `\xstacks_aftergroup:N` on group level 6+ will be restored at group level 5... anything between level 3 until the other mark will be restored at level 2.
>
> ***Note:*** if `\xstacks_aftergroup:N` is called at the same level (or above) of the mark, it will be equivalent to a simple `\group_insert_after:N`.
>
> ***Attention:*** These should be considered *deprecated*. They are less effective and flexible than `\groupmark_new:n` (from package *`tokgroupmark`*).

3 Multiple mark points after group variant

`\xstacks_groupmark:N`
`\xstacks_aftergroup:NN`

`\xstacks_groupmark:N {⟨int-var⟩}`
`\xstacks_aftergroup:NN {⟨int-var⟩} {⟨token⟩}`

⟨`int-var`⟩ must be an already declared integer variable, and will be used to mark/track a group level. That way it is possible to have multiple and independent return points. Otherwise it works exactly as the previous pair of commands. All assignments made to ⟨`int-var`⟩ are also local.

> ***Attention:*** These should be considered *deprecated*. They are less effective and flexible than `\groupmark_new:n` (from package *`tokgroupmark`*).

4 Custom Group Mark Commands

This is defined in the package *`tokgroupmark`*. The obvious distinction with the previous commands: this don't depend on the user reserving an integer for it (as in `\xstacks_groupmark:N`) and, unlike `\xstacks_groupmark:`, it allows for multiple sets (no commonalities).

`\groupmark_new:n`

`\groupmark_new:n {⟨mark-prefix⟩}`

This will globally create two commands (below), named after ⟨`mark-prefix`⟩, to "mark" and "restore" a token up to that "mark".

> ***Note:*** Internally, an unique integer will be created to track the group mark.
>
> ***Note:*** An error will be raised if ⟨`mark-prefix`⟩ is already used.

`\<mark-prefix>_groupmark:`
`\<mark-prefix>_aftergroup:N`

`\<mark-prefix>_groupmark:`
`\<mark-prefix>_aftergroup:N {⟨token⟩}`

The `\<mark-prefix>_groupmark:` will "mark" (save) the current group level, so that, when using `\<mark-prefix>_aftergroup:N` the token will be restored once the same group level is reached once again (similar to `\aftergroup`).

> ***Note:*** `\<mark-prefix>_groupmark:` will save (local assignment) the current group in an (unique to the set) integer. Since the assignment is local, it is possible to have more than one mark associated with the same set.

5 Stack command variant

These are defined in the package *tokglobalstack*.

`\globalstack_csnew:n`

```
\globalstack_csnew:n {⟨stack-prefix⟩}
```

This will globally create a set of commands, named after ⟨stack-prefix⟩, to push, put and pop items from a private global stack. All assignments to/from that stack will be global, and the stack itself will be unique to the command's set.

> ***Attention:*** This package defines `\xstacks_cs_gset:N` and `\xstacks_cs_gset:n` as aliases to this, for continuity, as it was originally defined in this package.

`\<stack-prefix>_gpush:n`
`\<stack-prefix>_gput_right:n`
`\<stack-prefix>_gput_left:n`
`\<stack-prefix>_gpop:`

```
\<stack-prefix>_gpush:n {⟨tokens⟩}
\<stack-prefix>_gput_right:n {⟨tokens⟩}
\<stack-prefix>_gput_left:n {⟨tokens⟩}
\<stack-prefix>_gput_gpop:
```

The `\<stack-prefix>_gpush:n` will push ⟨tokens⟩ (can be any number of tokens) into a global, private, stack. `\<stack-prefix>_gput_right:n` and `\<stack-prefix>_gput_left:n` will amend tokens to it, and `\<stack-prefix>_gpop:`, as the name implies, will insert the top of the stack into the input stream. That way it is possible to have a very fine control of what, where and when the items are collected and used.

6 Stack variable variant

These are also defined in the package *tokglobalstack*.

`\globalstack_gset:N`
`\globalstack_gpush:Nn`
`\globalstack_gput_right:Nn`
`\globalstack_gput_left:Nn`
`\globalstack_gpop:N`

```
\globalstack_gset:N {⟨stack-var⟩}
\globalstack_gpush:Nn {⟨stack-var⟩} {⟨tokens⟩}
\globalstack_gput_right:Nn {⟨stack-var⟩} {⟨tokens⟩}
\globalstack_gput_left:Nn {⟨stack-var⟩} {⟨tokens⟩}
\globalstack_gpop:N {⟨stack-var⟩}
```

`\globalstack_gset:N` will globally create a stack variable named ⟨stack-var⟩ (a specialized token list variable). Once created it is possible to push tokens into it (`\globalstack_gpush:Nn`), amend tokens to the top (`\globalstack_gput_right:Nn` and `\globalstack_gput_left:Nn`) and pop those tokens (`\globalstack_gpop:N`) into the input stream. All assignments being global.

> ***Attention:*** This package defines a series of aliases, for definition continuity, as follow:
> `\xstacks_gset:N` → `\globalstack_new:N`
> `\xstacks_gpush:Nn` → `\globalstack_gpush:Nn`
> `\xstacks_gput_right:Nn` → `\globalstack_gput_right:Nn`
> `\xstacks_gput_left:Nn` → `\globalstack_gput_left:Nn`
> `\xstacks_gpop:N` → `\globalstack_gpop:N`

7 Benchmarks and Final Thoughts

In the following, there is an exercise of . . . , better said, to evaluate the advantage/disadvantage of each approach, in an extreme case: multiple tokens, deeply nested groups. The most effective strategy is the stack command variant (see 5): *40 ops* (7.4) versus *160* (7.3). In lighter cases, taking in account that the stack variants are mostly 'constant time' (they don't depend on how deep the grouping is, but just how many operations (push/pop) are needed), the difference diminishes, but the stack command variant still comes first (*20 ops* versus *35 ops*).

At the end, it's a case of flexibility, convenience and programming style versus performance. In case of the original problem, to save context past the end of a scope, the stack approach is the fastest, but the after group variant doesn't lags much behind, while behaving much like the `\aftergroup` primite: tokens are promptly restored past the end of a local group (at the target group).

> ***Note:*** About the after group variant, if `\int_if_compare:nNnTF` is used (instead of the primitive `\if_int_compare:w`), the number of *ops* almost triple! Take a look at the code, the version with `\int_if_compare:nNnTF` is commented out.

> ***Note:*** Not really needed, but since one is at it, you can try (for more stable results) `\fp_set:Nn \g_benchmark_duration_target_fp {20}`. . .

7.1 Single mark, after group variant

```
\benchmark:n
  {
    \group_begin:
      \xstacks_groupmark:
    {{
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpc_bool
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpc_bool
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpc_bool
    {{
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpb_bool
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpb_bool
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpb_bool
    {{
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpa_bool
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpa_bool
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpa_bool
    {{
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpa_bool
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpa_bool
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpa_bool
    {{
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpa_bool
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpa_bool
        \xstacks_aftergroup:N \bool_set_true:N
        \xstacks_aftergroup:N \l__mytest_tmpa_bool
    }} }} }}
    }} }}
    \group_end:
  }
```

On average, it took about *160 ops* (*430 ops* if using `\int_if_compare:nNnTF`). If 'only' the 4 first groups, the average goes down to just *35 ops* (*90 ops* if using `\int_if_compare:nNnTF`). Obviously, the number of after groups raises exponentially, 2^n, with the number of nested groups.

Attention: Again, these commands are to be considered *deprecated*. They are less effective and flexible than `\groupmark_new:n` (from package *tokgroupmark*).

7.2 Multiple marks, after group variant

```
    \int_gzero_new:N \myMark_int
\benchmark:n
  {
    \group_begin:
      \xstacks_groupmark:N \myMark_int
    {{
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpc_bool
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpc_bool
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpc_bool
    {{
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpb_bool
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpb_bool
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpb_bool
    {{
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpa_bool
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpa_bool
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpa_bool
    {{
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpa_bool
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpa_bool
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpa_bool
    {{
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpa_bool
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpa_bool
        \xstacks_aftergroup:NN \myMark_int \bool_set_true:N
        \xstacks_aftergroup:NN \myMark_int \l__mytest_tmpa_bool
    }} }} }}
    }} }}
    \group_end:
  }
```

On average, it took about *200 ops* (*480 ops* if using `\int_if_compare:nNnTF`). If ‘only’ the 4 first groups, the average goes down to just *45 ops* (*95 ops* if using `\int_if_compare:nNnTF`). Likewise, the number of after groups raises exponentially, 2^n, with the number of nested groups (more expensive than the previous one because of the extra integer that has to be carried on).

Attention: Again, these commands are to be considered *deprecated*. They are less effective and flexible than `\groupmark_new:n` (from package `tokgroupmark`).

7.3 Custom group mark commands variant

```
    \groupmark_new:n {myMark}
\benchmark:n
  {
    \group_begin:
      \myMark_groupmark:
    {{
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpc_bool
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpc_bool
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpc_bool
    {{
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpb_bool
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpb_bool
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpb_bool
    {{
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpa_bool
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpa_bool
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpa_bool
    {{
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpa_bool
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpa_bool
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpa_bool
    {{
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpa_bool
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpa_bool
        \myMark_aftergroup:N \bool_set_true:N
        \myMark_aftergroup:N \l__mytest_tmpa_bool
    }} }} }}
    }} }}
    \group_end:
  }
```

On average, it took about *160 ops* (*430 ops* if using `\int_if_compare:nNnTF`). If ‘only’ the 4 first groups, the average goes down to just *35 ops* (*90 ops* if using `\int_if_compare:nNnTF`). Obviously, the number of after groups raises exponentially, 2^n, with the number of nested groups.

This has, naturally, the same performance/behaviour as 7.1, but with the flexibility of 7.2.

7.4 Stack command variant

```
    \globalstack_csnew:n {myStack}
\benchmark:n
  {
    \group_begin:
    {{
        \myStack_gpush:n {
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
        }
    {{
        \myStack_gput_right:n {
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
        }
    {{
        \myStack_gput_right:n {
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
        }
    {{
        \myStack_gput_right:n {
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
        }
    {{
        \myStack_gput_right:n {
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
        }
    }} }} }}
    }} }}
        \myStack_gpop:
    \group_end:
  }
```

On average, it took about *40 ops*. If ‘only’ the 4 first groups, the average goes down to about *22 ops*. All operations, `\myStack_gpush:n`, `\myStack_gput_right:n` and `\myStack_gpop:` are, more or less, equally expensive.

The original implementation (version 1.0a) took, on average, about *405 ops*. If ‘only’ the 4 first groups, the average was about *195 ops*. What’s the difference? Using just `\int_use:N` instead of `\int_to_Alph:n` when creating internal variables (with `\csname`) to save the tokens.

7.5 Stack variable variant

```
    \globalstack_new:N \g_mytest_stack
\benchmark:n
  {
    \group_begin:
    {{
        \globalstack_gpush:Nn \g_mytest_stack {
          \bool_set_true:N \l__mytest_tmpc_bool
          \bool_set_true:N \l__mytest_tmpc_bool
          \bool_set_true:N \l__mytest_tmpc_bool
        }
    {{
        \globalstack_gput_right:Nn \g_mytest_stack {
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
          \bool_set_true:N \l__mytest_tmpb_bool
        }
    {{
        \globalstack_gput_right:Nn \g_mytest_stack {
          \bool_set_true:N \l__mytest_tmpc_bool
          \bool_set_true:N \l__mytest_tmpc_bool
          \bool_set_true:N \l__mytest_tmpc_bool
        }
    {{
        \globalstack_gput_right:Nn \g_mytest_stack {
          \bool_set_true:N \l__mytest_tmpc_bool
          \bool_set_true:N \l__mytest_tmpc_bool
          \bool_set_true:N \l__mytest_tmpc_bool
        }
    {{
        \globalstack_gput_right:Nn \g_mytest_stack {
          \bool_set_true:N \l__mytest_tmpc_bool
          \bool_set_true:N \l__mytest_tmpc_bool
          \bool_set_true:N \l__mytest_tmpc_bool
        }
    }} }} }}
    }} }}
        \globalstack_gpop:N   \g_mytest_stack
    \group_end:
  }
```

On average, it took about *50 ops*. If ‘only’ the 4 first groups, the average goes down to about *30 ops*. The `\globalstack_gpush:Nn` and `\globalstack_gpop:N` are somewhat the more expensive operations.

The original implementation (version 1.0a) took, on average, about *310 ops*. If ‘only’ the 4 first groups, the average remained about *290 ops*. What’s the difference? Using just `\int_use:N` instead of `\int_to_Alph:n` when creating internal variables (with `\csname`) to save the tokens.

References

[1] David Carlisle. *Stackexchange about grouping*. 2026. URL: https://tex.stackexchange.com/questions/757755/coffins-scope-groups#comment1889872_757755 (visited on 01/01/2026).

[2] Alceu Frigeri. *The tokglobalstack package*. 2026. URL: https://ctan.org/pkg/tokglobalstack (visited on 02/18/2026).

[3] Alceu Frigeri. *The tokgroupmark package*. 2026. URL: https://ctan.org/pkg/tokgroupmark (visited on 02/18/2026).