

The `withargs` package*

Michiel Helvensteijn
mhelvens+latex@gmail.com

September 7, 2015

Development of this package is organized at github.com/mhelvens/latex-withargs.
I am happy to receive feedback there!

1 Introduction

An example is worth a thousand words:

```
\withargs [!] [Hello] [world] { #2 #3#1 }
```

```
Hello world!
```

A quick explanation: we're passing three pieces of L^AT_EX code in the form of optional arguments to the final argument, which forms the output. We're defining a new anonymous macro and invoking it right away. Up to seven optional arguments are supported.

1.1 Why is this useful?

One of the main use-cases for this package is to insert the expanded result of a computation into the middle of a big blob of code that *shouldn't* be expanded. This is possible because argument substitution bypasses the expansion process:

```
\def \expectedResult {\bar}  
\def \actualResult {\bas}  
\withargs (oo) [\actualResult] [\expectedResult] {  
  \texttt{\detokenize{  
    The \foo variable resulted in '#1' instead of '#2'!  
  }}  
}
```

```
The \foo variable resulted in '\bas ' instead of '\bar '!
```

'(oo)' indicates that both arguments should be expanded 'once'. Any optional L^AT_EX3 style argument specification between parentheses may be specified to control expansion.

*This document corresponds to `withargs` v0.1.0, dated 2015/09/07.

You can define your own commands accepting ‘templates’ with T_EX-style parameters:

```

\newcommand{\NewPost}[4]{
  % #1: author      % #2: title
  % #3: content     % #4: template
  \withargs (xnn) [#1] [#2] [#3] {#4}
}

\def\MyPost{ \NewPost{Author}{Title}
  {Boy, do I have some important stuff to say!} }

\begin{tabular}{p{3.5cm}p{3.5cm}p{3.5cm}}
  \MyPost{\textbf{#2\hfill#1}\vskip1mm\hrule\vskip1mm\textit{#3}} &
  \MyPost{\fbox{#1: ``#2''}\par#3\vskip1mm\hrule} &
  \MyPost{#2: #3\hfill\textit{(#1)}} &
\end{tabular}

```

<u>Title</u>	<u>Author</u>	<u>Author: “Title”</u>	Title: Boy, do I have
<i>Boy, do I have some important stuff to say!</i>	Boy, do I have some important stuff to say!	Boy, do I have some important stuff to say!	some important stuff to say! (Author)

Generally, if speed is not a concern, `\withargs` can be used to make L^AT_EX code more readable in a variety of situations.

2 Document Level Commands

```

\withargs  $\overbrace{(\langle argument\ specification \rangle)}^x$   $\overbrace{[\langle 1 \rangle] [\langle 2 \rangle] [\langle 3 \rangle] [\langle 4 \rangle] [\langle 5 \rangle] [\langle 6 \rangle] [\langle 7 \rangle] \{\langle body \rangle\}}^x$ 

```

Leaves *body* in the output with #1...#7 replaced by optional arguments *1*...*7*.

An optional L^AT_EX3 style *argument specification* between parentheses can be provided, in which case the arguments undergo expansion as specified before being placed in the *body*. Here is an example demonstrating the possible expansion types:

```

\def\foo{\bar} \def\s{s} \def\bar{ba\s} \def\bas{FOO-BAR-BAS}

\withargs (n o f x c v )
  [\foo] [\foo] [\foo] [\foo] [\foo] [\foo] {
  \begin{tabular}{llll}
    n: & \texttt{\detokenize{#1}} & & % no expansion
    o: & \texttt{\detokenize{#2}} & \backslash & % expand once
    f: & \texttt{\detokenize{#3}} & & % expand until unexpandable
    x: & \texttt{\detokenize{#4}} & \backslash & % exhaustive expansion
    c: & \texttt{\detokenize{#5}} & & % \csname conversion
    v: & \texttt{\detokenize{#6}} & & % 'c', then 'o'
  \end{tabular}
}

```

n:	\foo	o:	\bar
f:	ba\s	x:	bas
c:	\bas	v:	FOO-BAR-BAS

Note that all spaces in the *argument specification* are ignored. For details about L^AT_EX3 argument specifications, have a look at the following documentation:

<http://www.ctan.org/pkg/exp13>

`\uniquecsname` Perhaps a bit misleading, `\uniquecsname` is not actually defined as a command, but `\withargs` recognizes it as a special marker. If an optional `\withargs` argument consists entirely of `\uniquecsname`, it is replaced by a command sequence name which is guaranteed to be unique... unless you look at the source code (Section 4) and intentionally replicate the naming scheme.

<pre> \withargs (ccc) [\uniquecsname][\uniquecsname][\uniquecsname] { \def#1{A} \let\A#1 \def#2{B} \def#3{C} #3#2#1% } % (\A) </pre>
<pre>CBA(A)</pre>

3 L^AT_EX3 Functions

The L^AT_EX3 functions underlying the functionality of `\withargs` are also available as is.

`\withargs:nnnnnnnn` $\overbrace{\{\langle 1 \rangle\} \{\langle 2 \rangle\} \{\langle 3 \rangle\} \{\langle 4 \rangle\} \{\langle 5 \rangle\} \{\langle 6 \rangle\} \{\langle 7 \rangle\} \{\langle 8 \rangle\} \{\langle body \rangle\}}$

These functions do pretty much the same thing as the main `\withargs` macro, except that the argument specification is embedded in the function name as per L^AT_EX3 coding convention, so a parameter slot is freed up for custom use. They are slightly faster than `\withargs`, as there is no need to gather optional arguments or do any error checking. The downside is that the `\uniquecsname` marker doesn't work for these.

To use a specific expansion scheme, you have to define a variant:

<pre> \ExplSyntaxOn \cs_generate_variant:Nn \withargs:nn {cn} \withargs:cn {LaTeX} {#1} \ExplSyntaxOff </pre>
<pre>L^AT_EX</pre>

Read the L^AT_EX3 documentation for details.

4 Implementation

We now show and explain the entire implementation from `withargs.sty`.

4.1 Package Info

```
1 \NeedsTeXFormat{LaTeX2e}
2 \RequirePackage{expl3}
3 \ProvidesExplPackage{withargs}{2015/09/07}{0.1.0}
4 {an inline construct for passing token lists as TeX parameters}
```

4.2 Required Packages

We just need some `expl3`-ish packages.

```
5 \RequirePackage{xparse}
6 \RequirePackage{l3regex}
```

4.3 L^AT_EX3 Functions

```
\withargs:nm  $\overbrace{\langle 1 \rangle \langle 2 \rangle \langle 3 \rangle \langle 4 \rangle \langle 5 \rangle \langle 6 \rangle \langle 7 \rangle \langle 8 \rangle}^{1-8}$  \langle body \rangle
\withargs:nnn
\withargs:nnnn
\withargs:nnnnn
\withargs:nnnnnn
\withargs:nnnnnnn
\withargs:nnnnnnnn
\withargs:nnnnnnnnn
```

These are the `expl3` API versions of the `\withargs` command. The implementation is quite straight-forward. This technique has to be used by any library command that accepts T_EX-style parameters.

```
7 \cs_new_protected:Nn \withargs:nm {
8   \cs_set:Npn \__withargs:n ##1 {#2}
9     \__withargs:n {#1} }
10 \cs_new_protected:Nn \withargs:nnn {
11   \cs_set:Npn \__withargs:nn ##1##2 {#3}
12     \__withargs:nn {#1}{#2} }
13 \cs_new_protected:Nn \withargs:nnnn {
14   \cs_set:Npn \__withargs:nnn ##1##2##3 {#4}
15     \__withargs:nnn {#1}{#2}{#3} }
16 \cs_new_protected:Nn \withargs:nnnnn {
17   \cs_set:Npn \__withargs:nnnn ##1##2##3##4 {#5}
18     \__withargs:nnnn {#1}{#2}{#3}{#4} }
19 \cs_new_protected:Nn \withargs:nnnnnn {
20   \cs_set:Npn \__withargs:nnnnn ##1##2##3##4##5 {#6}
21     \__withargs:nnnnn {#1}{#2}{#3}{#4}{#5} }
22 \cs_new_protected:Nn \withargs:nnnnnnn {
23   \cs_set:Npn \__withargs:nnnnnn ##1##2##3##4##5##6 {#7}
24     \__withargs:nnnnnn {#1}{#2}{#3}{#4}{#5}{#6} }
25 \cs_new_protected:Nn \withargs:nnnnnnnn {
26   \cs_set:Npn \__withargs:nnnnnnn ##1##2##3##4##5##6##7 {#8}
27     \__withargs:nnnnnnn {#1}{#2}{#3}{#4}{#5}{#6}{#7} }
28 \cs_new_protected:Nn \withargs:nnnnnnnnn {
29   \cs_set:Npn \__withargs:nnnnnnnn ##1##2##3##4##5##6##7##8 {#9}
30     \__withargs:nnnnnnnn {#1}{#2}{#3}{#4}{#5}{#6}{#7}{#8} }
```

4.4 Document Level Command

`_withargs_var:x` $\{\langle argument\ specifier\rangle\}$

This is a convenience command for generating and using a `\withargs:` variant in one go. I only use it for the document-level command, since those users can't roll their own.

`#1` should be the number of optional `\withargs` arguments and `#2` should be a \LaTeX 3 argument specification not longer than `#1` — a prefix.

```
31 \cs_new:Nn \_withargs_var:nx {
32   \cs_generate_variant:cx
33     { withargs : \prg_replicate:nn{#1}{n} n }
34     { #2 n }
35   \use:c {
36     withargs :
37     #2
38     \prg_replicate:nn{#1-\tl_count:n{#2}}{n}
39     n
40   }
41 }
```

```
42 \cs_generate_variant:Nn \cs_generate_variant:Nn {cx}
```

`_withargs_process_uniqecname:n` $\{\langle argument\rangle\}$

An `xparse` processor function to pass a unique control sequence name if the argument given was `'\uniquecname'`.

```
43 \cs_new_protected:Nn \_withargs_process_uniqecname:n{
44   \tl_if_eq:nnTF {#1} {\uniquecname} {
45     \int_gincr:N \g__with_unique_int
46     \tl_set:Nx \ProcessedArgument
47       { Unique-CS-Name-( \int_use:N\g__with_unique_int ) }
48   }{
49     \tl_set:Nn \ProcessedArgument {#1}
50   }
51 }
52 \int_new:N \g__with_unique_int
```

`_withargs_remove_spaces:n` $\{\langle argument\rangle\}$

An `xparse` processor function to remove all spaces from the argument.

```
53 \cs_new_protected:Nn \_withargs_remove_spaces:n{
54   \tl_set:Nn \ProcessedArgument {#1}
55   \tl_remove_all:Nn\ProcessedArgument {~}
56 }
```

`\withargs` (*⟨argument specification⟩*) [*⟨1⟩*] [*⟨2⟩*] [*⟨3⟩*] [*⟨4⟩*] [*⟨5⟩*] [*⟨6⟩*] [*⟨7⟩*] {*⟨body⟩*}

This is the document version of the `\withargs` command.

```

57 \NewDocumentCommand {\withargs}
58   { >{\__withargs_remove_spaces:n}          D(){} % argument spec
59     >{\__withargs_process_uniquesname:n} +o    % up to 7 optional args
60     >{\__withargs_process_uniquesname:n} +o
61     >{\__withargs_process_uniquesname:n} +o
62     >{\__withargs_process_uniquesname:n} +o
63     >{\__withargs_process_uniquesname:n} +o
64     >{\__withargs_process_uniquesname:n} +o
65     >{\__withargs_process_uniquesname:n} +o
66                                     +m } { % the body to execute

```

We first check if the argument specification is valid. It has to be between 0 and 7 characters long and each symbol has to be one of ‘noxfcv’. Otherwise: error! The variants ‘N’ and ‘V’ are not supported (yet) because they collect arguments differently than the others, and frankly, I didn’t want to bother.

```

67 \regex_match:nnF {^[noxfcv]{0,7}$} {#1}
68   { \msg_critical:nnn{withargs}{invalid-parameter-specs}{#1} }

```

The next bit counts the number of optional arguments provided using binary search. If `#1` specifies *more* arguments than were provided: error!

```

69 \int_set:Nn \l__with_arg_int {
70   \IfNoValueTF {#5}
71     { \IfNoValueTF {#3} { \IfNoValueTF{#2} 0 1 }
72       { \IfNoValueTF{#4} 2 3 } }
73     { \IfNoValueTF {#7} { \IfNoValueTF{#6} 4 5 }
74       { \IfNoValueTF{#8} 6 7 } }
75   }
76
77 \int_compare:nNnT {\tl_count:n{#1}} > {\l__with_arg_int} {
78   \msg_error:nnxxx{withargs}{invalid-parameter-number}
79     { \tl_count:n{#1} }
80     { \int_use:N \l__with_arg_int }
81     { #1 }
82   }

```

We can then call the right variant of `\withargs`:

```

83 \int_case:nnF {\l__with_arg_int} {
84   {1} { \__withargs_var:nx1{#1} {#2}           {#9} }
85   {2} { \__withargs_var:nx2{#1} {#2}{#3}      {#9} }
86   {3} { \__withargs_var:nx3{#1} {#2}{#3}{#4}  {#9} }
87   {4} { \__withargs_var:nx4{#1} {#2}{#3}{#4}{#5} {#9} }
88   {5} { \__withargs_var:nx5{#1} {#2}{#3}{#4}{#5}{#6} {#9} }
89   {6} { \__withargs_var:nx6{#1} {#2}{#3}{#4}{#5}{#6}{#7} {#9} }
90   {7} { \__withargs_var:nx7{#1} {#2}{#3}{#4}{#5}{#6}{#7}{#8}{#9} }
91   {}

```

```

92 }
93 \int_new:N \l__with_arg_int

```

The following is the error message displayed if the argument specification is ill-formed:

```

94 \msg_new:nnnn{withargs}{invalid-parameter-specs}{
95   The~argument~specification~'#1'~is~not~valid.
96 }{
97   The~argument~specification~should~consist~of~between~one~
98   and~seven~of~the~letters~'n',~'o',~'f',~'x',~'c',~'v'.
99 }

```

This is the error message displayed if the number of provided optional arguments is inconsistent with the provided argument specification.

```

100 \msg_new:nnnn{withargs}{invalid-parameter-number}{
101   You~specified~#1~arguments~but~provided~#2.
102 }{
103   Your~argument~specification~is~'#3',~which~means~you~should~
104   provide~#1~optional~arguments.~However,~you~provided~#2.~
105   You~should~fix~that.
106 }

```

Change History

0.0.1		optional and delimited by parentheses	6
General: initial version	1		
0.0.2	0.1.0	General: adjusted code to new version	
General: renamed package to withargs	1	of expl3	1
\withargs: made the first argument op-			

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		
__withargs:n	8, 9	__withargs:nnnnnnnn
__withargs:nn	11, 12	__withargs_process_uniquecname:n
__withargs:nnn	14, 15	43, 43, 59, 60, 61, 62, 63, 64, 65
__withargs:nnnn	17, 18	__withargs_remove_spaces:n
__withargs:nnnnn	20, 21	53, 53, 58
__withargs:nnnnnn	23, 24	__withargs_var:nx
__withargs:nnnnnnn	26, 27	31, 84, 85, 86, 87, 88, 89, 90
		__withargs_var:x
		31

C		P	
<code>\cs_generate_variant:cx</code>	32	<code>\prg_replicate:nn</code>	33, 38
<code>\cs_generate_variant:Nn</code>	42	<code>\ProcessedArgument</code>	46, 49, 54, 55
<code>\cs_new:Nn</code>	31	<code>\ProvidesExplPackage</code>	3
<code>\cs_new_protected:Nn</code>		R	
..	7, 10, 13, 16, 19, 22, 25, 28, 43, 53	<code>\regex_match:nnF</code>	67
<code>\cs_set:Npn</code> ...	8, 11, 14, 17, 20, 23, 26, 29	<code>\RequirePackage</code>	2, 5, 6
G		T	
<code>\g__with_unique_int</code>	45, 47, 52	<code>\tl_count:n</code>	38, 77, 79
I		<code>\tl_if_eq:nnTF</code>	44
<code>\IfNoValueTF</code>	70, 71, 72, 73, 74	<code>\tl_remove_all:Nn</code>	55
<code>\int_case:nnF</code>	83	<code>\tl_set:Nn</code>	49, 54
<code>\int_compare:nNnT</code>	77	<code>\tl_set:Nx</code>	46
<code>\int_gincr:N</code>	45	U	
<code>\int_new:N</code>	52, 93	<code>\uniquecsname</code>	3, 44
<code>\int_set:Nn</code>	69	<code>\use:c</code>	35
<code>\int_use:N</code>	47, 80	W	
L		<code>\withargs</code>	2, 57, 57
<code>\l__with_arg_int</code>	69, 77, 80, 83, 93	<code>\withargs:mn</code>	7, 7
M		<code>\withargs:nnn</code>	7, 10
<code>\msg_critical:nnn</code>	68	<code>\withargs:nnnn</code>	7, 13
<code>\msg_error:nnxxx</code>	78	<code>\withargs:nnnnn</code>	7, 16
<code>\msg_new:nnnn</code>	94, 100	<code>\withargs:nnnnnn</code>	7, 19
N		<code>\withargs:nnnnnnn</code>	7, 22
<code>\NeedsTeXFormat</code>	1	<code>\withargs:nnnnnnnn</code>	7, 25
<code>\NewDocumentCommand</code>	57	<code>\withargs:nnnnnnnnn</code>	7, 28