Hook management for commands*

Frank Mittelbach Phelype Oleinik

October 30, 2026

1 Introduction

This file implements generic hooks for (arbitrary) commands. In theory every command \⟨*name*⟩ offers now two associated hooks to which code can be added using `\AddToHook`,[1] `\AddToHookNext`, `\AddToHookWithArguments`, and `\AddToHookNextWithArguments`.[2]

However, this is only true "in theory". In practice there are a number of restrictions that makes it impossible to use such generic command hooks in a number of cases, so please read all of section 2 to understand what may prevent you from using them successfully.

The generic command hooks are:

`cmd/`⟨*name*⟩`/before` This hook is executed at the very start of the command, right after its arguments (if any) are parsed. The hook ⟨*code*⟩ runs in the command inside a call to `\UseHookWithArguments`. Any code added to this hook using `\AddToHookWithArguments` or `\AddToHookNextWithArguments` can access the command's arguments using `#1`, `#2`, etc., up to the number of arguments of the command. If `\AddToHook` or `\AddToHookNext` are used, the arguments cannot be accessed (see the lthooks documentation[3] on hooks with arguments).

`cmd/`⟨*name*⟩`/after` This hook is similar to `cmd/`⟨*name*⟩`/before`, but it is executed at the very end of the command body. This hook is implemented as a reversed hook.

The hooks are not physically present before `\begin{document}`[4] (i.e., using a command in the preamble will never execute the hook) and if nobody has declared any code for them, then they are not added to the command code ever. For example, if we have the following definition

```
\newcommand\foo[2]{Code #1 for #2!}
```

then executing `\foo{A}{B}` will simply run `Code␣A␣for␣B!` as it was always the case. However, if somebody, somewhere (e.g., in a package) adds

*This file has version v1.0n dated 2026-06-01, © LaTeX Project.

[1] In this documentation, when something is being said about `\AddToHook`, the same will be valid for `\AddToHookWithArguments`, unless that particular paragraph is highlighting the differences between both. The same is true for the other hook-related functions and their `...WithArguments` counterparts.

[2] In practice this is not supported for all types of commands, see section 2.4 for the restrictions that apply and what happens if one tries to use this with commands for which this is not supported.

[3] `texdoc lthooks-doc`

[4] More specifically, they are inserted in the commands after the `begindocument` hook, so they are also not present while LaTeX is reading the `.aux` file.

```
\AddToHook{cmd/foo/before}{<before code>}
```

then, after `\begin{document}` the definition of `\foo` will be:

```
\renewcommand\foo[2]{%
  \UseHookWithArguments{cmd/foo/before}{2}{#1}{#2}%
  Code #1 for #2!}
```

and similarly `\AddToHook{cmd/foo/after}{<after code>}` alters the definition to

```
\renewcommand\foo[2]{%
  Code #1 for #2!%
  \UseHookWithArguments{cmd/foo/after}{2}{#1}{#2}}
```

In other words, the mechanism is similar to what etoolbox offers with `\pretocmd` and `\apptocmd` with the important differences

- that code can be prepended or appended (i.e., added to the hooks) even if the command itself is not (yet) defined, because the defining package has not been loaded at this point;
- and that by using the hook management interface it is now possible to define how the code chunks added in these places are ordered, if different packages want to add code at these points.

2 Restrictions and operational details

Adding arbitrary material to commands is tricky because most of the time we do not know what the macro expects as arguments when expanding and TEX doesn't have a reliable way to see that, so some guesswork has to be employed.

We can do this in most cases when commands are defined using `\NewDocumentCommand` or `\newcommand` (with a few exceptions). For commands defined with `\def` the situation is less good. Common cases where the command hooks will not work are:

- Commands that use special catcode settings within their definition. In that case it is usually not possible to augment the definition (see 2.1).
- If a command is defined while `\ExplSyntaxOn` is in force **and** the command contains `~` characters to represent spaces, then it can't be patched to include the command hooks. In fact in some very special circumstances you might even get a low-level error rather than the information that the command can't be patched (see, for example, https://github.com/latex3/latex2e/issues/1430.
- Commands that have arguments as far as the user is concerned (e.g., `\section` or `\caption`), but are defined in a way that these arguments are not read by the user level command but only later during the processing. In that case the `after` hook doesn't work at all. The before hook only works with `\AddToHook` but not with `\AddToHookWithArguments` because the arguments haven't been read at that point where the hook is patched in. See section 2.4.
- Adding a specific generic command hook is only attempted once per command, thus after redefining a command such hooks will no longer be there and will also not being re-added, see section 2.1.1.

All this means that you have to have a good understanding of how commands are defined when you attempt to make use of such hooks and something goes wrong. What can help in that case is to turn on `\DebugHooksOn` in which case you get much more (low-level) details on why something fails and what was tried to enable the hooks.

2.1 Patching

The code here tries to find out if a command was defined with `\newcommand` or `\DeclareRobustCommand` or `\NewDocumentCommand`, and if so it *assumes* that the argument specification of the command is as expected (which is not fail-proof, if someone redefines the internals of these commands in devious ways, but is a reasonable assumption).

If the command is one of the defined types, the code here does a sandboxed expansion of the command such that it can be redefined again exactly as before, but with the hook code added.

If however the command is not a known type (it was defined with `\def`, for example), then the code uses an approach similar to etoolbox's `\patchcmd` to retokenize the command with the hook code in place. This procedure, however, is more likely to fail if the catcode settings are not the same as the ones at the time of command's definition, so not always adding a hook to a command will work.

2.1.1 Timing

When `\AddToHook` (or its expl3 equivalent) is called with a generic `cmd` hook, say, `cmd/foo/before`, for the first time (that is, no code was added to that same hook before), in the preamble of a document, it will store a patch instruction for that command until `\begin{document}`, and only then all the commands which had hooks added will be patched in one go. That means that no command in the preamble will have hooks patched into them.

At `\begin{document}` all the delayed patches will be executed, and if the command doesn't exist the code is still added to the hook, but it will not be executed. After `\begin{document}`, when `\AddToHook` is called with a generic `cmd` hook the first time, the command will be immediately patched to include the hook, and if it doesn't exist or if it can't be patched for any reason, an error is thrown; if `\AddToHook` was already used in the preamble no new patching is attempted.

This has the consequence that a command defined or redefined after `\begin{document}` only uses generic `cmd` hook code if `\AddToHook` is called for the first time after the definition is made, or if the command explicitly uses the generic hook in its definition by declaring it with `\NewHookPair` adding `\UseHook` as part of the code.[5]

2.2 Command copies

Once a hook has been added to a command, it will be present in any copies of that command which are made. For example, if in the preamble we have

```
\NewDocumentCommand\foo{}{}
\AddToHook{cmd/foo/after}{1}%
```

then after `\begin{document}`, any use of `\NewCommandCopy` will include the hook in the copy, for example

[5] We might change this behavior in the main document slightly after gaining some usage experience.

```
\NewCommandCopy\baz\foo
```

would include the hook in this copied command.

2.3 Grouping

Command hooks are intended to be added to document commands, which are typically defined globally. As such, adding a hook is a global operation, even if the command was previously only locally-defined. Addition of material to hooks is also global.

2.4 Commands that look ahead

Some commands are defined in different "steps" and they look ahead in the input stream to find more arguments. If you try to add some code to the `cmd/`⟨*name*⟩`/after` hook of such command, it will not work, and it is not possible to detect that programmatically, so the user has to know (or find out) which commands can or cannot have hooks attached to them.

One good example is the `\section` command. You can add something to the `cmd/section/before` hook (but only with `\AddToHook` not `\AddToHookWithArguments`), but if you try to add anything to the `cmd/section/after` hook, `\section` will no longer work at all. That happens because the `\section` macro takes no argument, but instead calls a few internal LaTeX macros to look for the optional and mandatory arguments. By adding code to the `cmd/section/after` hook, you get in the way of that scanning.

In such a case, where it is known that a specific generic command hook does not work if code is added to it, the package author can add a `\DisableGenericHook`[6] declaration to prevent this from happening in user documents and thereby avoiding obscure errors.

3 Package author interface

The `cmd` hooks are, by default, available for all commands that can be patched to add the hooks. For some commands, however, the very beginning or the very end of the code is not the best place to put the hooks, for example, if the command looks ahead for arguments (see section 2.4).

If you are a package author and you want to add the hooks to your own commands in the proper position you can define the command and manually add the `\UseHookWithArguments` calls inside the command in the proper positions, and manually define the hooks with `\NewHookWithArguments` or `\NewReversedHookWithArguments`. When the hooks are explicitly defined, patching is not attempted so you can make sure your command works properly. For example, an (admittedly not really useful) command that typesets its contents in a framed box with width optionally given in parentheses:

```
\newcommand\fancybox{\@ifnextchar({\@fancybox}{\@fancybox(5cm)}}
\def\@fancybox(#1)#2{\fbox{\parbox{#1}{#2}}}
```

If you try that definition, then add some code after it with

```
\AddToHook{cmd/fancybox/after}{<code>}
```

[6] Please use `\DisableGenericHook` if at all, only on hooks that you "own", i.e., for commands your package or class defines and not second guess whether or not hooks of other packages should get disabled!

and then use the `\fancybox` command you will see that it will be completely broken, because the hook will get executed in the middle of parsing for optional `(...)` argument.

If, on the other hand, you want to add hooks to your command you can do something like:

```
\newcommand\fancybox{\@ifnextchar({\@fancybox}{\@fancybox(5cm)}}
\def\@fancybox(#1)#2{\fbox{%
               \UseHookWithArguments{cmd/fancybox/before}{2}{#1}{#2}%
               \parbox{#1}{#2}%
               \UseHookWithArguments{cmd/fancybox/after}{2}{#1}{#2}}}
\NewHookWithArguments{cmd/fancybox/before}{2}
\NewReversedHookWithArguments{cmd/fancybox/after}{2}
```

then the hooks will be executed where they should and no patching will be attempted. It is important that the hooks are declared with `\NewHookWithArguments` or `\NewReversedHookWithArguments`, otherwise the command hook code will try to patch the command. Note also that the call to `\UseHookWithArguments{cmd/fancybox/before}` does not need to be in the definition of `\fancybox`, but anywhere it makes sense to insert it (in this case in the internal `\@fancybox`).

Alternatively, if for whatever reason your command does not support the generic hooks provided here, you can disable a hook with `\DisableGenericHook`[7], so that when someone tries to add code to it they will get an error. Or if you don't want the error, you can simply declare the hook with `\NewHook` and never use it.

The above approach is useful for really complex commands where for one or the other reason the hooks can't be placed at the very beginning and end of the command body and some hand-crafting is needed. However, in the example above the real (and in fact only) issue is the cascading argument parsing in the style developed long ago in LaTeX 2.09. Thus, a much simpler solution for this case is to replace it with the modern `\NewDocumentCommand` syntax and define the command as follows:

```
\DeclareDocumentCommand\fancybox{D(){5cm}m}{\fbox{\parbox{#1}{#2}}}
```

If you do that then both hooks automatically work and are patched into the right places.

3.1 Arguments and redefining commands

The code in ltcmdhooks does its best to find out how many arguments a given command has, and to insert the appropriate call to `\UseHookWithArguments`, so that the arguments seen by the hook are exactly those grabbed by the command (the hook, after all, is a macro call, so the arguments have to be placed in the right order, or they won't match).

When using the package writer interface, as discussed in section 3, to change the position of the hooks in your commands, you are also free to change how the hook code in your command sees its arguments. When a `cmd` hook is declared with `\NewHook` (or `\NewHookWithArguments` or other variations of that), it loses its "generic" nature and works as a regular hook. This means that you may choose to declare it without arguments regardless if the command takes arguments or not, or declare it with arguments, even if the command takes none.

However, this flexibility should not be abused. When using a nonstandard configuration for the hook arguments, think reasonably: a user will expect that the argument

[7] Please use `\DisableGenericHook` if at all, only on hooks that you "own", i.e., for commands your package or class defines and not second guess whether or not hooks of other packages should get disabled!

`#1` in the hook corresponds to the argument's first argument, and so on. Any other configuration is likely to cause confusion and, if used, will have to be well documented.

This flexibility, however, allows you to "correct" the arguments for the hooks. For example, LaTeX's `\refstepcounter` has a single argument, the name of the counter. The cleveref package adds an optional argument to `\refstepcounter`, making the name of the counter argument `#2`. If the author of cleveref wanted, for whatever reason, to add hooks to `\refstepcounter`, to preserve compatibility he could write something along the lines of:

```
\NewHookWithArguments{cmd/refstepcounter/before}{1}
\renewcommand\refstepcounter[2][<default>]{%
  \UseHookWithArguments{cmd/refstepcounter/before}{1}{#2}%
  <code for \refstepcounter>}
```

so that the mandatory argument, which is arg `#2` in the definition, would still be seen as `#1` in the hook code.

Another possibility would be to place the optional argument as the second argument for the hook, so that people looking for it would be able to use it. In either case, it would have to be well documented to cause as little confusion as possible.

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.

A

`\AddToHook` 3
`\AddToHookWithArguments` 4

B

`begindocument` (hook) 1

C

`\caption` 2
`cmd` (hook) 4, 5
`cmd/⟨name⟩/after` (hook) 1, 4
`cmd/⟨name⟩/before` (hook) 1, 1
`cmd/foo/before` (hook) 3
`cmd/section/after` (hook) 4, 4
`cmd/section/before` (hook) 4

D

`\DebugHooksOn` 3
`\DisableGenericHook` 4

E

`\ExplSyntaxOn` 2

H

Hooks:
  `begindocument` 1
`cmd` 4, 5
`cmd/⟨name⟩/after` 1, 4
`cmd/⟨name⟩/before` 1, 1
`cmd/foo/before` 3
`cmd/section/after` 4, 4
`cmd/section/before` 4

N

`\newcommand` 2
`\NewCommandCopy` 3
`\NewDocumentCommand` 5
`\NewHook` 5
`\NewHookPair` 3
`\NewHookWithArguments` 5
`\NewReversedHookWithArguments` 5

R

`\refstepcounter` 6

S

`\section` 2

T

TeX and LaTeX 2_ε commands:
  `\AddToHook` 1
  `\AddToHookNext` 1

\AddToHookNextWithArguments 1
\AddToHookWithArguments 1
\apptocmd 2
\DeclareRobustCommand 3
\def 3
\newcommand 3
\NewDocumentCommand 3
\patchcmd 3
\pretocmd 2
\section 4

U

\UseHook 3
\UseHookWithArguments 1