The `ltcmdhooks` module[*]

Frank Mittelbach   Phelype Oleinik

November 30, 2022

1 Introduction

This file implements generic hooks for (arbitrary) commands. In theory every command $\backslash\langle name\rangle$ offers now two associated hooks to which code can be added using `\AddToHook` or `\AddToHookNext`.[1] These are

cmd/⟨*name*⟩/before This hook is executed at the very start of the command execution after its arguments (if any) are parsed. The hook ⟨*code*⟩ is wrapped in the command inside a call to `\UseHook{cmd/`⟨*name*⟩`/before}`, so the arguments passed to the command are *not* available in the hook ⟨*code*⟩.

cmd/⟨*name*⟩/after This hook is similar to cmd/⟨*name*⟩/after, 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}` (i.e., using a command in the preamble will never execute them) 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

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

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

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

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

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

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 defined, because the defining package has not yet been loaded

- 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.

---

[*]This file has version v1.0f dated 2021/10/20, © LATEX Project.

[1]In practice this is not supported for all types of commands, see section 2.2 for the restrictions that apply and what happens if one tries to use this with commands for which this is not supported.

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.

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.[2]

2.2 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.

---

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

One good example is the `\section` command. You can add something to the `cmd/section/before` hook, but if you try to add something to the `cmd/section/after` hook, `\section` will no longer work. 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.

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 2.2).

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 `\UseHook` calls inside the command in the proper positions, and manually define the hooks with `\NewHook` or `\NewReversedHook`. 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>}
```

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{%
                      \UseHook{cmd/fancybox/before}%
                      \parbox{#1}{#2}%
                      \UseHook{cmd/fancybox/after}}}
\NewHook{cmd/fancybox/before}
\NewReversedHook{cmd/fancybox/after}
```

then the hooks will be executed where they should and no patching will be attempted. It is important that the hooks are declared with `\NewHook` or `\NewReversedHook`, otherwise the command hook code will try to patch the command. Note also that the call to `\UseHook{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 `\DisableHook`[3], so that when someone

---

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

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.

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.