The `ltfilehook` package*

Frank Mittelbach

July 21, 2020

Contents

1 Introduction

1.1 Provided hooks

The code offers a number of hooks into which packages (or the user) can add code to support different use cases. Many hooks are offered as pairs (i.e., the second hook is reversed. Also important to know is that these pairs are properly nested with respect to other pairs of hooks.

There are hooks that are executed for all files of a certain type (if they contain code), e.g., for all "include files" or all "packages", and there are also hooks that are specific to a single file, e.g., do something after the package `foo.sty` has been loaded.

1.2 General hooks for file reading

There are four hooks that are called for each file that is read using document-level commands such as `\input`, `\include`, `\usepackage`, etc. They are not called for files read using internal low-level methods, such as `\@input` or `\openin`.

*This package has version v0.9a dated 2020/07/19, © LaTeX Project.

file/before
file/before/...
file/after/...
file/after

These are:

file/before, file/before/⟨*file-name*⟩ These hooks are executed in that order just before the file is loaded for reading. The code of the first hook is used is used with every file, while the second is executed only for the file with matching ⟨*file-name*⟩ allowing you to specify code that only applies to one file.

file/after/⟨*file-name*⟩, file/after These hooks are after the file with name ⟨*file-name*⟩ has been fully consumed. The order is swapped (the specific one comes first) so that the `before` and `after` hooks nest properly, which is important if any of them involve grouping (e.g., contain environments, for example). furthermore both hooks are reversed hooks to support correct nesting of different packages add code to both `/before` and `/after` hooks.

So the overall sequence of hook processing for any file read through the user interface commands of LaTeX is:

```
\UseHook{⟨file/before⟩}
\UseHook{⟨file/before/⟨file name⟩⟩}
   ⟨file contents⟩
\UseHook{⟨file/after/⟨file name⟩⟩}
\UseHook{⟨file/after⟩}
```

Todo: With a higher-level interface that doesn't matter, but it is a bit weird, if you use `\AddToHook` *or* `\hook_gput:nnn` *directly, so I guess that has to be done differently!*

The file hooks only refer to the file by its name and extension, so the ⟨*file name*⟩ should be the file name as it is on the filesystem with extension (if any) and without paths. Different from `\input` and similar commands, for hooks the `.tex` extension is not assumed, so `.tex` files must also be given with their extension. Files within subfolders should also be addressed by their name and extension only.

Extensionless files also work, and should then be given without extension. Note however that TeX prioritizes `.tex` files, so if two files `foo` and `foo.tex` exist in the search path, only the latter will be seen.

When a file is input, the ⟨*file name*⟩ is available in `\CurrentFile`, which is then used when accessing the `file/before/`⟨*file name*⟩ and `file/after/`⟨*file name*⟩.

\CurrentFile

The name of the file about to be read (or just finished) is available to the hooks through `\CurrentFile` (there is no `expl3` name for it for now). The file is always provided with its extension, i.e., how it appears on your hard drive, but without any specified path to it. For example, `\input{sample}` and `\input{app/sample.tex}` would both have `\CurrentFile` being `sample.tex`.

\CurrentFilePath

The path to the current file (complement to \CurrentFile) is available in \CurrentFilePath if needed. The paths returned in \CurrentFilePath are only user paths, given through \input@path (or expl3's equivalent \l_file_search_path_seq) or by directly typing in the path in the \input command or equivalent. Files located by kpsewhich get the path added internally by the TEX implementation, so at the macro level it looks as if the file were in the current folder, so the path in \CurrentFilePath is empty in these cases (package and class files, mostly).

1.3 Hooks for package and class files

Commands to load package and class files (e.g., \usepackage, \RequirePackage, \LoadPackageWithOptions, etc.) offer the hooks from section 1.2 when they are used to load a package or class file, e.g., file/after/array.sty would be called after the array package got loaded. But as packages and classes form as special group of files, there are some additional hooks available that only apply when a package or class is loaded.

package/before
package/after
package/before/...
package/after/...
class/before
class/after
class/before/...
class/after/...

These are:

package/before, package/after These hooks are called for each package being loaded.

package/before/⟨*name*⟩, package/after/⟨*name*⟩ These hooks are additionally called if the package name is ⟨*name*⟩ (without extension).

class/before, class/after These hooks are called for each class being loaded.

class/before/⟨*name*⟩, class/after/⟨*name*⟩ These hooks are additionally called if the class name is ⟨*name*⟩ (without extension).

All /after hooks are implemented as reversed hooks.
The overall sequence of execution for \usepackage and friends is therefore:

```
\UseHook{⟨package/before⟩}
\UseHook{⟨package/before/⟨package name⟩⟩}
    \UseHook{⟨file/before⟩}
    \UseHook{⟨file/before/⟨package name⟩.sty⟩}
        ⟨package contents⟩
    \UseHook{⟨file/after/⟨package name⟩.sty⟩}
    \UseHook{⟨file/after⟩}
    code from \AtEndOfPackage if used inside the package
\UseHook{⟨package/after/⟨package name⟩⟩}
\UseHook{⟨package/after⟩}
```

and similar for class file loading, except that package/ is replaced by class/ and \AtEndOfPackage by \AtEndOfClass.

If a package or class is not loaded (or it was loaded before the hooks were set) none of the hooks are executed!

1.4 Hooks for \include files

To manage \include files, LaTeX issues a \clearpage before and after loading such a file. Depending on the use case one may want to execute code before or after these \clearpages especially for the one that is issued at the end.

Executing code before the final \clearpage, means that the code is processed while the last page of the included material is still under construction. Executing code after it means that all floats from inside the include file are placed (which might have added further pages) and the final page has finished.

Because of these different scenarios we offer hooks in three places.[1] None of the hooks are executed when an \include file is bypassed because of an \includeonly declaration. They are, however, all executed if LaTeX makes an attempt to load the \include file (even if it doesn't exist and all that happens is "No file ⟨*filename*⟩.tex").

include/before
include/before/...
include/end
include/end/...
include/after
include/after/...

These are:

include/before, include/before/⟨*name*⟩ These hooks are executed one after another after the initial \clearpage and after .aux file is changed to use ⟨*name*⟩.aux, but before the ⟨*name*⟩.tex file is loaded. In other words they are executed at the very beginning of the first page of the \include file.

include/end/⟨*name*⟩, include/end These hooks are executed (in that order) after LaTeX has stopped reading from the \include file, but before it has issued a \clearpage to output any deferred floats.

include/after/⟨*name*⟩, include/after These hooks are executed (in that order) after LaTeX has issued the \clearpage but before is has switched back writing to the main .aux file. Thus technically we are still inside the \include and if the hooks generate any further typeset material including anything that writes to the .aux file, then it would be considered part of the included material and bypassed if it is not loaded because of some \includeonly statement.[2]

1.5 High-level interfaces for LaTeX

We do not provide any high-level LaTeX commands (like filehook or scrlfile do) but think that for package writers the commands from for hook management are sufficient.

1.6 A sample package for structuring the log output

As an application we provide the package structuredlog that adds lines to the .log when a file is opened and closed for reading keeping track of nesting level es well. For example, for the current document it adds the lines

```
= (LEVEL 1 START) t1lmr.fd
= (LEVEL 1 STOP) t1lmr.fd
= (LEVEL 1 START) supp-pdf.mkii
```

[1] If you want to execute code before the first \clearpage there is no need to use a hook—you can write it directly in front of the \include.

[2] For that reason another \clearpage is executed after these hooks which normally does nothing, but starts a new page if further material got added this way.

```
= (LEVEL 1 STOP) supp-pdf.mkii
= (LEVEL 1 START) nameref.sty
== (LEVEL 2 START) refcount.sty
== (LEVEL 2 STOP) refcount.sty
== (LEVEL 2 START) gettitlestring.sty
== (LEVEL 2 STOP) gettitlestring.sty
= (LEVEL 1 STOP) nameref.sty
= (LEVEL 1 START) ltfilehook-doc.out
= (LEVEL 1 STOP) ltfilehook-doc.out
= (LEVEL 1 START) ltfilehook-doc.out
= (LEVEL 1 STOP) ltfilehook-doc.out
= (LEVEL 1 START) ltfilehook-doc.hd
= (LEVEL 1 STOP) ltfilehook-doc.hd
= (LEVEL 1 START) ltfilehook.dtx
== (LEVEL 2 START) ot1lmr.fd
== (LEVEL 2 STOP) ot1lmr.fd
== (LEVEL 2 START) omllmm.fd
== (LEVEL 2 STOP) omllmm.fd
== (LEVEL 2 START) omslmsy.fd
== (LEVEL 2 STOP) omslmsy.fd
== (LEVEL 2 START) omxlmex.fd
== (LEVEL 2 STOP) omxlmex.fd
== (LEVEL 2 START) umsa.fd
== (LEVEL 2 STOP) umsa.fd
== (LEVEL 2 START) umsb.fd
== (LEVEL 2 STOP) umsb.fd
== (LEVEL 2 START) ts1lmr.fd
== (LEVEL 2 STOP) ts1lmr.fd
== (LEVEL 2 START) t1lmss.fd
== (LEVEL 2 STOP) t1lmss.fd
= (LEVEL 1 STOP) ltfilehook.dtx
```

Thus if you inspect an issue in the `.log` it is easy to figure out in which file it occurred, simply by searching back for `LEVEL` and if it is a `STOP` then remove 1 from the level value and search further for `LEVEL` with that value which should then be the `START` level of the file you are in.

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.