# The `ltfilehook` package*

Frank Mittelbach, Phelype Oleinik, LaTeX Project Team

March 15, 2022

## Contents

---

*This package has version v1.0m dated 2022/03/10, © LaTeX Project.

# 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`.

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

These are:

**`file/before, file/⟨file-name⟩/before`** These hooks are executed in that order just before the file is loaded for reading. The code of the first hook 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/⟨file-name⟩/after, 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 adding 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/⟨file name⟩/before⟩}
    ⟨file contents⟩
\UseHook{⟨file/⟨file name⟩/after⟩}
\UseHook{⟨file/after⟩}
```

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, the `.tex` extension is not assumed in hook ⟨*file name*⟩, so `.tex` files must be specified with their extension to be recognized. 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/⟨file name⟩/before` and `file/⟨file name⟩/after`.

**\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).

**\CurrentFileUsed**
**\CurrentFilePathUsed**  In normal circumstances these are identical to \CurrentFile and \CurrentFilePath. They will differ when a file substitution has occurred for \CurrentFile. In that case, \CurrentFileUsed and \CurrentFilePathUsed will hold the actual file name and path loaded by LaTeX, while \CurrentFile and \CurrentFilePath will hold the names that were *asked for*. Unless doing very specific work on the file being read, \CurrentFile and \CurrentFilePath should be enough.

## 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/array.sty/after 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/⟨name⟩/before, package/⟨name⟩/after**  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/⟨name⟩/before, class/⟨name⟩/after**  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/⟨package name⟩/before⟩}

        \UseHook{⟨file/before⟩}

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

*code from* \AtEndOfPackage *if used inside the package*

```
\UseHook{⟨package/⟨package name⟩/after⟩}
\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!

All class or package hooks involving the name of the class or package are implemented as one-time hooks, whereas all other such hooks are normal hooks. This allows for the following use case

```
\AddToHook{package/varioref/after}
    { ... apply my customizations if the package gets
      loaded (or was loaded already) ... }
```

without the need to first test if the package is already loaded.

## 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`").

---

[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.

| | |
|---|---|
| `include/before` | These are: |
| `include/.../before` | |
| `include/end` | |
| `include/.../end` | |
| `include/after` | |
| `include/.../after` | |

**`include/before, include/`⟨*name*⟩`/before`** These hooks are executed (in that order) 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/`⟨*name*⟩`/end, 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/`⟨*name*⟩`/after, 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]

All `include` hooks involving the name of the included file are implemented as one-time hooks (whereas all other such hooks are normal hooks).

## 1.5  High-level interfaces for LATEX

We do not provide any additional wrappers around the hooks (like filehook or scrlfile do) because we believe that for package writers the high-level commands from the hook management, e.g., `\AddToHook`, etc. are sufficient and in fact easier to work with, given that the hooks have consistent naming conventions.

## 1.6  Internal interfaces for LATEX

| | |
|---|---|
| `\declare@file@substitution` | `\declare@file@substitution   {`⟨*file*⟩`} {`⟨*replacement-file*⟩`}` |
| `\undeclare@file@substitution` | `\undeclare@file@substitution {`⟨*file*⟩`}` |

If ⟨*file*⟩ is requested for loading replace it with ⟨*replacement-file*⟩. `\CurrentFile` remains pointing to ⟨*file*⟩ but `\CurrentFileUsed` will show the file actually loaded.

The main use case for this declaration is to provide a corrected version of a package that can't be changed (due to its license) but no longer functions because of LATEX kernel changes, for example, or to provide a version that makes use of new kernel functionality while the original package remains available for use with older releases.

The `\undeclare@file@substitution` declaration undoes a substitution made earlier.

*Please do not misuse this functionality and replace a file with another unless if really needed and only if the new version is implementing the same functionality as the original one!*

---

[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.

\disable@package@load  {⟨*package*⟩}  {⟨*alternate-code*⟩}
\reenable@package@load {⟨*package*⟩}

If ⟨*package*⟩ is requested do not load it but instead run ⟨*alternate-code*⟩ which could issue a warning, error or any other code.

The main use case is for classes that want to restrict the set of supported packages or contain code that make the use of some packages impossible. So rather than waiting until the document breaks they can set up informative messages why certain packages are not available.

The function is only implemented for packages not for arbitrary files.

## 1.7   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
= (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.

# 2  The Implementation

1 ⟨∗2ekernel⟩

2 ⟨@@=filehook⟩

## 2.1  Document and package-level commands

`\CurrentFile`
`\CurrentFilePath`
`\CurrentFileUsed`
`\CurrentFilePathUsed`

User-level macros that hold the current file name and file path. These are used internally as well because the code takes care to protect against a possible redefinition of these macros in the loaded file (it's necessary anyway to make hooks work with nested `\input`). The versions `\...Used` hold the *actual* file name and path that is loaded by LaTeX, whereas the other two hold the name as requested. They will differ in case there's a file substitution.

3 ⟨/2ekernel⟩
4 ⟨∗2ekernel | latexrelease⟩
5 ⟨latexrelease⟩\*IncludeInRelease{2020/10/01}*%
6 ⟨latexrelease⟩                    *{\CurrentFile}{Hook management file}*%
7 \ExplSyntaxOn
8 \tl_new:N \CurrentFile
9 \tl_new:N \CurrentFilePath
10 \tl_new:N \CurrentFileUsed
11 \tl_new:N \CurrentFilePathUsed
12 \ExplSyntaxOff
13 ⟨/2ekernel | latexrelease⟩
14 ⟨latexrelease⟩\EndIncludeInRelease

15 ⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
16 ⟨latexrelease⟩                    {\CurrentFile}{Hook management file}%
17 ⟨latexrelease⟩
18 ⟨latexrelease⟩\let \CurrentFile         \@undefined
19 ⟨latexrelease⟩\let \CurrentFilePath     \@undefined
20 ⟨latexrelease⟩\let \CurrentFileUsed     \@undefined
21 ⟨latexrelease⟩\let \CurrentFilePathUsed \@undefined
22 ⟨latexrelease⟩
23 ⟨latexrelease⟩\EndIncludeInRelease
24 ⟨∗2ekernel⟩

(*End definition for* `\CurrentFile` *and others. These functions are documented on page* *3.*)

## 2.2  **expl3** helpers

25 ⟨/2ekernel⟩
26 ⟨∗2ekernel | latexrelease⟩
27 ⟨latexrelease⟩\*IncludeInRelease{2020/10/01}*%
28 ⟨latexrelease⟩                    *{\__filehook_file_parse_full_name:nN}{File helpers}*%
29 \ExplSyntaxOn

`\_filehook_file_parse_full_name:nN`
`\__filehook_full_name:nn`

A utility macro to trigger expl3's file-parsing and lookup, and return a normalized representation of the file name. If the queried file doesn't exist, no normalization takes place.

The output of \__filehook_file_parse_full_name:nN is passed on to the #2—a 3-argument macro that takes the ⟨*path*⟩, ⟨*base*⟩, and ⟨*ext*⟩ parts of the file name.

```
30 \cs_new:Npn \__filehook_file_parse_full_name:nN #1
31   {
32     \exp_args:Nf \file_parse_full_name_apply:nN
33       {
34         \exp_args:Nf \__filehook_full_name:nn
35           { \file_full_name:n {#1} } {#1}
36       }
37   }
38 \cs_new:Npn \__filehook_full_name:nn #1 #2
39   {
40     \tl_if_empty:nTF {#1}
41       { \tl_trim_spaces:n {#2} }
42       { \tl_trim_spaces:n {#1} }
43   }
```

(*End definition for* \__filehook_file_parse_full_name:nN *and* \__filehook_full_name:nn.)

\__filehook_if_no_extension:nTF  
\__filehook_drop_extension:N

Some actions depend on whether the file extension was explicitly given, and sometimes the extension has to be removed. The macros below use \__filehook_file_parse_-full_name:nN to split up the file name and either check if ⟨*ext*⟩ (#3) is empty, or discard it.

```
44 \cs_new:Npn \__filehook_if_no_extension:nTF #1
45   {
46     \exp_args:Ne \tl_if_empty:nTF
47       { \file_parse_full_name_apply:nN {#1} \use_iii:nnn }
48   }
49 \cs_new_protected:Npn \__filehook_drop_extension:N #1
50   {
51     \tl_gset:Nx #1
52       {
53         \exp_args:NV \__filehook_file_parse_full_name:nN #1
54           \__filehook_drop_extension_aux:nnn
55       }
56   }
57 \cs_new:Npn \__filehook_drop_extension_aux:nnn #1 #2 #3
58     { \tl_if_empty:nF {#1} { #1 / } #2 }
```

(*End definition for* \__filehook_if_no_extension:nTF *and* \__filehook_drop_extension:N.)

\g__filehook_input_file_seq  
\l__filehook_internal_tl  
\__filehook_file_push:  
\__filehook_file_pop:  
\__filehook_file_pop_assign:nnnn

Yet another stack, to keep track of \CurrentFile and \CurrentFilePath with nested \inputs. At the beginning of \InputIfFileExists, the current value of \CurrentFilePath and \CurrentFile is pushed to \g__filehook_input_file_seq, and at the end, it is popped and the value reassigned. Some other places don't use \InputIfFileExists directly (\include) or need \CurrentFile earlier (\@onefilewithoptions), so these are manually used elsewhere as well.

```
59 \tl_new:N \l__filehook_internal_tl
60 \seq_if_exist:NF \g__filehook_input_file_seq
61   { \seq_new:N \g__filehook_input_file_seq }
62 \cs_new_protected:Npn \__filehook_file_push:
63   {
64     \seq_gpush:Nx \g__filehook_input_file_seq
```

```
65          {
66              { \CurrentFilePathUsed } { \CurrentFileUsed }
67              { \CurrentFilePath     } { \CurrentFile     }
68          }
69      }
70  \cs_new_protected:Npn \__filehook_file_pop:
71      {
72          \seq_gpop:NNTF \g__filehook_input_file_seq \l__filehook_internal_tl
73              { \exp_after:wN \__filehook_file_pop_assign:nnnn \l__filehook_internal_tl }
74              {
75                  \msg_error:nnn { latex2e } { should-not-happen }
76                      { Tried~to~pop~from~an~empty~file~name~stack. }
77              }
78      }
79  \cs_new_protected:Npn \__filehook_file_pop_assign:nnnn #1 #2 #3 #4
80      {
81          \tl_set:Nn \CurrentFilePathUsed {#1}
82          \tl_set:Nn \CurrentFileUsed {#2}
83          \tl_set:Nn \CurrentFilePath {#3}
84          \tl_set:Nn \CurrentFile {#4}
85      }
86  \ExplSyntaxOff
```

(*End definition for* `\g__filehook_input_file_seq` *and others.*)

```
87  ⟨/2ekernel | latexrelease⟩
88  ⟨latexrelease⟩\EndIncludeInRelease
```

When rolling forward the following expl3 functions may not be defined. If we roll back the code does nothing.

```
89  ⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
90  ⟨latexrelease⟩                          {\file_parse_full_name_apply:nN}{Roll forward help}%
91  ⟨latexrelease⟩
92  ⟨latexrelease⟩\ExplSyntaxOn
93  ⟨latexrelease⟩\cs_if_exist:NF\file_parse_full_name_apply:nN
94  ⟨latexrelease⟩{
95  ⟨latexrelease⟩\cs_new:Npn \file_parse_full_name_apply:nN #1
96  ⟨latexrelease⟩  {
97  ⟨latexrelease⟩     \exp_args:Ne \__file_parse_full_name_auxi:nN
98  ⟨latexrelease⟩        { \__kernel_file_name_sanitize:n {#1} }
99  ⟨latexrelease⟩  }
100 ⟨latexrelease⟩\cs_new:Npn \__file_parse_full_name_auxi:nN #1
101 ⟨latexrelease⟩  {
102 ⟨latexrelease⟩     \__file_parse_full_name_area:nw { } #1
103 ⟨latexrelease⟩      / \s__file_stop
104 ⟨latexrelease⟩  }
105 ⟨latexrelease⟩\cs_new:Npn \__file_parse_full_name_area:nw #1 #2 / #3 \s__file_stop
106 ⟨latexrelease⟩  {
107 ⟨latexrelease⟩     \tl_if_empty:nTF {#3}
108 ⟨latexrelease⟩        { \__file_parse_full_name_base:nw { } #2 . \s__file_stop {#1} }
109 ⟨latexrelease⟩        { \__file_parse_full_name_area:nw { #1 / #2 }
110 ⟨latexrelease⟩                                         #3 \s__file_stop }
111 ⟨latexrelease⟩  }
112 ⟨latexrelease⟩\cs_new:Npn \__file_parse_full_name_base:nw #1 #2 . #3 \s__file_stop
113 ⟨latexrelease⟩  {
```

```
114 ⟨latexrelease⟩    \tl_if_empty:nTF {#3}
115 ⟨latexrelease⟩      {
116 ⟨latexrelease⟩        \tl_if_empty:nTF {#1}
117 ⟨latexrelease⟩          {
118 ⟨latexrelease⟩            \tl_if_empty:nTF {#2}
119 ⟨latexrelease⟩              { \__file_parse_full_name_tidy:nnnN { } { } }
120 ⟨latexrelease⟩              { \__file_parse_full_name_tidy:nnnN { .#2 } { } }
121 ⟨latexrelease⟩          }
122 ⟨latexrelease⟩          { \__file_parse_full_name_tidy:nnnN {#1} { .#2 } }
123 ⟨latexrelease⟩      }
124 ⟨latexrelease⟩      { \__file_parse_full_name_base:nw { #1 . #2 }
125 ⟨latexrelease⟩                                          #3 \s__file_stop }
126 ⟨latexrelease⟩  }
127 ⟨latexrelease⟩\cs_new:Npn \__file_parse_full_name_tidy:nnnN #1 #2 #3 #4
128 ⟨latexrelease⟩  {
129 ⟨latexrelease⟩    \exp_args:Nee #4
130 ⟨latexrelease⟩      {
131 ⟨latexrelease⟩        \str_if_eq:nnF {#3} { / } { \use_none:n }
132 ⟨latexrelease⟩        #3 \prg_do_nothing:
133 ⟨latexrelease⟩      }
134 ⟨latexrelease⟩      { \use_none:n #1 \prg_do_nothing: }
135 ⟨latexrelease⟩      {#2}
136 ⟨latexrelease⟩  }
137 ⟨latexrelease⟩}
138 ⟨latexrelease⟩\ExplSyntaxOff
139 ⟨latexrelease⟩
140 ⟨latexrelease⟩\EndIncludeInRelease
141 ⟨∗2ekernel⟩

142 ⟨@@=⟩
```

## 2.3   Declaring the file-related hooks

These hooks have names with three-parts that start with file/, include/, class/ or
package/ and end with /before or /after (or /end in the case of include/). They are
all generic hooks so will be declared only if code is added to them; this declaration is
done for you automatically and, indeed, they should not be declared explicitly.

Those named .../after and include/.../end are, when code is added, declared
as reversed hooks.

## 2.4   Patching LaTeX's \InputIfFileExists command

Most of what we have to do is adding \UseHook into several LaTeX $2_\varepsilon$ core commands,
because of some circular dependencies in the kernel we do this only now and not in
ltfiles.

\InputIfFileExists  \InputIfFileExists loads any file if it is available so we have to add the hooks
\@input@file@exists@with@hooks  file/before and file/after in the right places. If the file doesn't exist no hooks
\unqu@tefilef@und  should be executed.

```
143 ⟨/2ekernel⟩
144 ⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
145 ⟨latexrelease⟩                  {\InputIfFileExists}{Hook management (files)}%
146 ⟨∗2ekernel | latexrelease⟩
```

```
147  \let\InputIfFileExists\@undefined
148  \DeclareRobustCommand \InputIfFileExists[2]{%
149    \IfFileExists{#1}%
150      {%
151        \@expl@@@filehook@file@push@@
152        \@filehook@set@CurrentFile
```

We pre-expand `\@filef@und` so that in case another file is loaded in the true branch of `\InputIfFileExists`, these don't change their value meanwhile. This isn't a worry with `\CurrentFile...` because they are kept in a stack.

```
153        \expandafter\@swaptwoargs\expandafter
154          {\expandafter\@input@file@exists@with@hooks
155            \expandafter{\@filef@und}}%
156          {#2}%
157        \@expl@@@filehook@file@pop@@
158      }%
159  }
160  \def\@input@file@exists@with@hooks#1{%
```

If the file exists then `\CurrentFile` holds its name. But we can't rely on that still being true after the file has been processed. Thus for using the name in the file hooks we need to preserve the name and then restore it for the `file/.../after` hook.

The hook always refers to the file requested by the user. The hook is *always* loaded for `\CurrentFile` which usually is the same as `\CurrentFileUsed`. In the case of a file replacement, the `\CurrentFileUsed` holds the actual file loaded. In any case the file names are normalized so that the hooks work on the real file name, rather than what the user typed in.

expl3's `\file_full_name:n` normalizes the file name (to factor out differences in the `.tex` extension), and then does a file lookup to take into account a possible path from `\l_file_search_path_seq` and `\input@path`. However only the file name and extension are returned so that file hooks can refer to the file by their name only. The path to the file is returned in `\CurrentFilePath`.

```
161    \edef\reserved@a{%
162      \@expl@@@filehook@file@pop@assign@@nnnn
163        {\CurrentFilePathUsed}%
164        {\CurrentFileUsed}%
165        {\CurrentFilePath}%
166        {\CurrentFile}}%
167    \expandafter\@swaptwoargs\expandafter{\reserved@a}%
```

Before adding to the file list we need to make all (letter) characters catcode 11, because several packages use constructions like

```
\filename@parse{<filename>}
\ifx\filename@ext\@clsextension
  ...
\fi
```

and that doesn't work if `\filename@ext` is `\detokenize`d. Making `\@clsextension` a string doesn't help much because some packages define their own `\<prefix>@someextension` with normal catcodes. This is not entirely correct because packages loaded (somehow) with catcode 12 alphabetic tokens (say, as the result of a `\string` or `\detokenize` command, or from a TeX string like `\jobname`) will have these character tokens incorrectly

11

turned into letter tokens. This however is rare, so we'll go for the all-letters approach (grepping the packages in TeX Live didn't bring up any obvious candidate for breaking with this catcode change).

```
168        {\edef\reserved@a{\unqu@tefilef@und#1\@nil}%
169         \@addtofilelist{\string@makeletter\reserved@a}%
170         \UseHook{file/before}%
```

The current file name is available in `\CurrentFile` so we use that in the specific hook.

```
171         \UseHook{file/\CurrentFile/before}%
172         \@@input #1% <- trailing space comes from \@filef@und
173        }%
```

And here, `\CurrentFile` is restored (by `\@expl@@@filehook@file@pop@assign@@nnnn`) so we can use it once more.

```
174      \UseHook{file/\CurrentFile/after}%
175      \UseHook{file/after}}
176   \def\unqu@tefilef@und"#1" \@nil{#1}
```

Now declare the non-generic file hooks used above:

```
177   \NewHook{file/before}
178   \NewReversedHook{file/after}
179   ⟨latexrelease⟩\EndIncludeInRelease
180   ⟨/2ekernel | latexrelease⟩
```

Now define `\InputIfFileExists` to input #1 if it seems to exist. Immediately prior to the input, #2 is executed. If the file #1 does not exist, execute '#3'.

```
181   ⟨latexrelease⟩\IncludeInRelease{2019/10/01}%
182   ⟨latexrelease⟩              {\InputIfFileExists}{Hook management (files)}%
183   ⟨latexrelease⟩
184   ⟨latexrelease⟩\DeclareRobustCommand \InputIfFileExists[2]{%
185   ⟨latexrelease⟩  \IfFileExists{#1}%
186   ⟨latexrelease⟩      {%
187   ⟨latexrelease⟩  \expandafter\@swaptwoargs\expandafter
188   ⟨latexrelease⟩       {\@filef@und}{#2\@addtofilelist{#1}\@@input}}}
189   ⟨latexrelease⟩\let\@input@file@exists@with@hooks\@undefined
190   ⟨latexrelease⟩\let\unqu@tefilef@und\@undefined
191   ⟨latexrelease⟩\EndIncludeInRelease
192   ⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
193   ⟨latexrelease⟩              {\InputIfFileExists}{Hook management (files)}%
194   ⟨latexrelease⟩\long\def \InputIfFileExists#1#2{%
195   ⟨latexrelease⟩  \IfFileExists{#1}%
196   ⟨latexrelease⟩      {#2\@addtofilelist{#1}\@@input \@filef@und}}
```

Also undo the internal command as some packages unfortunately test for their existence instead of using `\IfFormatAtLeastTF`.

```
197   ⟨latexrelease⟩\expandafter\let\csname InputIfFileExists \endcsname\@undefined
198   ⟨latexrelease⟩\let\@input@file@exists@with@hooks\@undefined
199   ⟨latexrelease⟩\let\unqu@tefilef@und\@undefined
200   ⟨latexrelease⟩\EndIncludeInRelease
201   ⟨∗2ekernel⟩
```

(*End definition for* `\InputIfFileExists`*,* `\@input@file@exists@with@hooks`*, and* `\unqu@tefilef@und`*. These functions are documented on page* **??***.*)

## 2.5  Declaring a file substitution

<sub>202</sub> ⟨@@=filehook⟩

<sub>203</sub> ⟨/2ekernel⟩
<sub>204</sub> ⟨*2ekernel | latexrelease⟩
<sub>205</sub> ⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
<sub>206</sub> ⟨latexrelease⟩                {\__filehook_subst_add:nn}{Declaring file substitution}%
<sub>207</sub> \ExplSyntaxOn

\__filehook_subst_add:nn
\__filehook_subst_remove:n
\__filehook_subst_file_normalize:Nn
\__filehook_subst_empty_name_chk:NN

\__filehook_subst_add:nn declares a file substitution by doing a (global) definition of the form \def\@file-subst@⟨file⟩{⟨replacement⟩}. The file names are properly sanitised, and normalized with the same treatment done for the file hooks. That is, a file replacement is declared by using the file name (and extension, if any) only, and the file path should not be given. If a file name is empty it is replaced by .tex (the empty csname is used to check that).

```
208 \cs_new_protected:Npn \__filehook_subst_add:nn #1 #2
209   {
210     \group_begin:
211       \cs_set:cpx { } { \exp_not:o { \cs:w\cs_end: } }
212       \int_set:Nn \tex_escapechar:D { -1 }
213       \cs_gset:cpx
214         {
215           @file-subst@
216           \__filehook_subst_file_normalize:Nn \use_ii_iii:nnn {#1}
217         }
218         { \__filehook_subst_file_normalize:Nn \__filehook_file_name_compose:nnn
219                                       {#2} }
220     \group_end:
221   }
222 \cs_new_protected:Npn \__filehook_subst_remove:n #1
223   {
224     \group_begin:
225       \cs_set:cpx { } { \exp_not:o { \cs:w\cs_end: } }
226       \int_set:Nn \tex_escapechar:D { -1 }
227       \cs_undefine:c
228         {
229           @file-subst@
230           \__filehook_subst_file_normalize:Nn \use_ii_iii:nnn {#1}
231         }
232     \group_end:
233   }
234 \cs_new:Npn \__filehook_subst_file_normalize:Nn #1 #2
235   {
236     \exp_after:wN \__filehook_subst_empty_name_chk:NN
237       \cs:w \exp_after:wN \cs_end:
238         \cs:w \__filehook_file_parse_full_name:nN {#2} #1 \cs_end:
239   }
240 \cs_new:Npn \__filehook_subst_empty_name_chk:NN #1 #2
241   { \if_meaning:w #1 #2 .tex \else: \token_to_str:N #2 \fi: }
```

(*End definition for* \__filehook_subst_add:nn *and others.*)

\use_ii_iii:nnn    A variant of \use_... to discard the first of three arguments.

*Todo: this should move to expl3*

<sub>242</sub> `\cs_gset:Npn \use_ii_iii:nnn #1 #2 #3 {#2 #3}`

(*End definition for* `\use_ii_iii:nnn`.)

<sub>243</sub> `\ExplSyntaxOff`
<sub>244</sub> ⟨/2ekernel | latexrelease⟩
<sub>245</sub> ⟨latexrelease⟩`\EndIncludeInRelease`
<sub>246</sub> ⟨∗2ekernel⟩

<span style="color:red">`\declare@file@substitution`</span>
<span style="color:red">`\undeclare@file@substitution`</span>

For two internals we provide LaTeX $2_\varepsilon$ names so that we can use them elsewhere in the kernel (and so that they can be used in packages if really needed, e.g., scrlfile).

<sub>247</sub> ⟨/2ekernel⟩
<sub>248</sub> ⟨∗2ekernel | latexrelease⟩
<sub>249</sub> ⟨latexrelease⟩`\IncludeInRelease{2020/10/01}%`
<sub>250</sub> ⟨latexrelease⟩ `                {\declare@file@substitution}{File substitution}%`
<sub>251</sub> `\ExplSyntaxOn`
<sub>252</sub> `\cs_new_eq:NN \declare@file@substitution   \__filehook_subst_add:nn`
<sub>253</sub> `\cs_new_eq:NN \undeclare@file@substitution \__filehook_subst_remove:n`
<sub>254</sub> `\ExplSyntaxOff`
<sub>255</sub> ⟨/2ekernel | latexrelease⟩
<sub>256</sub> ⟨latexrelease⟩`\EndIncludeInRelease`

We are not fully rolling back the file substitutions in case a rollback encounters a package that contains them, but is itself not setup for rollback. So we just bypass them and hope for the best.

<sub>257</sub> ⟨latexrelease⟩`\IncludeInRelease{0000/00/00}%`
<sub>258</sub> ⟨latexrelease⟩ `                {\declare@file@substitution}{File substitution}%`
<sub>259</sub> ⟨latexrelease⟩
<sub>260</sub> ⟨latexrelease⟩`\let \declare@file@substitution   \@gobbletwo`
<sub>261</sub> ⟨latexrelease⟩`\let \undeclare@file@substitution \@gobble`
<sub>262</sub> ⟨latexrelease⟩
<sub>263</sub> ⟨latexrelease⟩`\EndIncludeInRelease`
<sub>264</sub> ⟨∗2ekernel⟩

(*End definition for* `\declare@file@substitution` *and* `\undeclare@file@substitution`. *These functions are documented on page 5.*)

<sub>265</sub> ⟨@@=⟩
<sub>266</sub> `\ExplSyntaxOff`

## 2.6 Selecting a file (`\set@curr@file`)

`\set@curr@file`
`\set@curr@file@nosearch`
`\@curr@file`
`\@curr@file@reqd`

Now we hook into `\set@curr@file` to resolve a possible file substitution, and add `\@expl@@@filehook@set@curr@file@@nNN` at the end, after `\@curr@file` is set.

A file name is built using `\expandafter\string\csname`⟨*filename*⟩`\endcsname` to avoid expanding utf8 active characters. The `\csname` expands the normalization machinery and the routine to resolve a file substitution, returning a control sequence with the same name as the file.

It happens that when ⟨*filename*⟩ is empty, the generated control sequence is `\csname\endcsname`, and doing `\string` on that results in the file `csnameendcsname.tex`. To guard against that we `\ifx`-compare the generated control sequence with the empty csname. To do so, `\csname\endcsname` has to be defined, otherwise it would be equal to `\relax` and we would have false positives. Here we define `\csname\endcsname` to expand to itself to avoid it matching the definition of some other control sequence.

<sub>267</sub> ⟨/2ekernel⟩

14

```
268 ⟨∗2ekernel | latexrelease⟩
269 ⟨latexrelease⟩\IncludeInRelease{2022/06/01}%
270 ⟨latexrelease⟩                  {\set@curr@file}{Setting current file name}%
271 \def\set@curr@file{%
272   \begingroup
273     \set@curr@file@aux}
274 \edef\set@curr@file@nosearch{%
275   \begingroup
276     \let\noexpand\input@path\noexpand\@empty
277     \csname seq_clear:N\endcsname
278       \expandafter\noexpand\csname l_file_search_path_seq\endcsname
279     \noexpand\set@curr@file@aux}
280 \def\set@curr@file@aux#1{%
281     \escapechar\m@ne
282     \let\protect\string
283     \edef~{\string~}%
284     \expandafter\def\csname\expandafter\endcsname
285       \expandafter{\csname\endcsname}%
```

Two file names are set here: `\@curr@file@reqd` which is the file requested by the user, and `\@curr@file` which should be the same, except when we have a file substitution, in which case it holds the actual loaded file. `\@curr@file` is resolved first, to check if a substitution happens. If it doesn't, `\@expl@@@filehook@if@file@replaced@@TF` short-cuts and just copies `\@curr@file`, otherwise the full normalization procedure is executed.

At this stage the file name is parsed and normalized, but if the input doesn't have an extension, the default `.tex` is *not* added to `\@curr@file` because for applications other than `\input` (graphics, for example) the default extension may not be `.tex`. First check if the input has an extension, then if the input had no extension, call `\@expl@@@filehook@drop@extension@@N`. In case of a file substitution, `\@curr@file` will have an extension.

```
286     \@expl@@@filehook@if@no@extension@@nTF{#1}%
287       {\@tempswatrue}{\@tempswafalse}%
288     \@kernel@make@file@csname\@curr@file
289       \@expl@@@filehook@resolve@file@subst@@w {#1}%
290     \@expl@@@filehook@if@file@replaced@@TF
291       {\@kernel@make@file@csname\@curr@file@reqd
292         \@expl@@@filehook@normalize@file@name@@w{#1}%
293        \if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file@reqd \fi}%
294       {\if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file \fi
295        \global\let\@curr@file@reqd\@curr@file}%
296     \@expl@@@filehook@clear@replacement@flag@@
297   \endgroup}
298 ⟨/2ekernel | latexrelease⟩
299 ⟨latexrelease⟩\EndIncludeInRelease

300 ⟨latexrelease⟩\IncludeInRelease{2021/06/01}%
301 ⟨latexrelease⟩                  {\set@curr@file}{Setting current file name}%
302 ⟨latexrelease⟩\def\set@curr@file#1{%
303 ⟨latexrelease⟩  \begingroup
304 ⟨latexrelease⟩    \escapechar\m@ne
305 ⟨latexrelease⟩    \let\protect\string
306 ⟨latexrelease⟩    \edef~{\string~}%
307 ⟨latexrelease⟩    \expandafter\def\csname\expandafter\endcsname
```

```
308 ⟨latexrelease⟩        \expandafter{\csname\endcsname}%
309 ⟨latexrelease⟩        \@expl@@@filehook@if@no@extension@@nTF{#1}%
310 ⟨latexrelease⟩          {\@tempswatrue}{\@tempswafalse}%
311 ⟨latexrelease⟩        \@kernel@make@file@csname\@curr@file
312 ⟨latexrelease⟩          \@expl@@@filehook@resolve@file@subst@@w {#1}%
313 ⟨latexrelease⟩        \@expl@@@filehook@if@file@replaced@@TF
314 ⟨latexrelease⟩          {\@kernel@make@file@csname\@curr@file@reqd
315 ⟨latexrelease⟩             \@expl@@@filehook@normalize@file@name@@w{#1}%
316 ⟨latexrelease⟩           \if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file@reqd \fi}%
317 ⟨latexrelease⟩          {\if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file \fi
318 ⟨latexrelease⟩           \global\let\@curr@file@reqd\@curr@file}%
319 ⟨latexrelease⟩        \@expl@@@filehook@clear@replacement@flag@@
320 ⟨latexrelease⟩    \endgroup}
321 ⟨latexrelease⟩\let\set@curr@file@nosearch\@undefined
322 ⟨latexrelease⟩\EndIncludeInRelease

323 ⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
324 ⟨latexrelease⟩                 {\set@curr@file}{Setting current file name}%
325 ⟨latexrelease⟩\def\set@curr@file#1{%
326 ⟨latexrelease⟩  \begingroup
327 ⟨latexrelease⟩      \escapechar\m@ne
328 ⟨latexrelease⟩      \expandafter\def\csname\expandafter\endcsname
329 ⟨latexrelease⟩        \expandafter{\csname\endcsname}%
330 ⟨latexrelease⟩      \@expl@@@filehook@if@no@extension@@nTF{#1}%
331 ⟨latexrelease⟩        {\@tempswatrue}{\@tempswafalse}%
332 ⟨latexrelease⟩      \@kernel@make@file@csname\@curr@file
333 ⟨latexrelease⟩        \@expl@@@filehook@resolve@file@subst@@w {#1}%
334 ⟨latexrelease⟩      \@expl@@@filehook@if@file@replaced@@TF
335 ⟨latexrelease⟩        {\@kernel@make@file@csname\@curr@file@reqd
336 ⟨latexrelease⟩           \@expl@@@filehook@normalize@file@name@@w{#1}%
337 ⟨latexrelease⟩         \if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file@reqd \fi}%
338 ⟨latexrelease⟩        {\if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file \fi
339 ⟨latexrelease⟩         \global\let\@curr@file@reqd\@curr@file}%
340 ⟨latexrelease⟩      \@expl@@@filehook@clear@replacement@flag@@
341 ⟨latexrelease⟩    \endgroup}
342 ⟨latexrelease⟩\let\set@curr@file@nosearch\@undefined
343 ⟨latexrelease⟩\EndIncludeInRelease

344 ⟨latexrelease⟩\IncludeInRelease{2019/10/01}%
345 ⟨latexrelease⟩                 {\set@curr@file}{Setting current file name}%
346 ⟨latexrelease⟩\def\set@curr@file#1{%
347 ⟨latexrelease⟩  \begingroup
348 ⟨latexrelease⟩      \escapechar\m@ne
349 ⟨latexrelease⟩      \xdef\@curr@file{%
350 ⟨latexrelease⟩        \expandafter\expandafter\expandafter\unquote@name
351 ⟨latexrelease⟩        \expandafter\expandafter\expandafter{%
352 ⟨latexrelease⟩        \expandafter\string
353 ⟨latexrelease⟩          \csname\@firstofone#1\@empty\endcsname}}%
354 ⟨latexrelease⟩  \endgroup
355 ⟨latexrelease⟩}
356 ⟨latexrelease⟩\let\set@curr@file@nosearch\@undefined
357 ⟨latexrelease⟩\EndIncludeInRelease

358 ⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
359 ⟨latexrelease⟩                 {\set@curr@file}{Setting current file name}%
360 ⟨latexrelease⟩\let\set@curr@file\@undefined
```

(*End definition for* \set@curr@file *and others. These functions are documented on page* **??**.)

**\@filehook@set@CurrentFile**
**\@kernel@make@file@csname**
**\@set@curr@file@aux**

*Todo: This should get internalized using* `@expl@` *names*

```
368 \def\@kernel@make@file@csname#1#2#3{%
369   \xdef#1{\expandafter\@set@curr@file@aux
370     \csname\expandafter#2\@firstofone#3\@nil\endcsname}}
```

This auxiliary compares \⟨*filename*⟩ with \csname\endcsname to check if the empty
.tex file was requested.

```
371 \def\@set@curr@file@aux#1{%
372   \expandafter\ifx\csname\endcsname#1%
373     .tex\else\string#1\fi}
```

Then we call `\@expl@@@filehook@set@curr@file@@nNN` once for `\@curr@file`
to set `\CurrentFile(Path)Used` and once for `\@curr@file@reqd` to set
`\CurrentFile(Path)`. Here too the slower route is only used if a substitution
happened, but here `\@expl@@@filehook@if@file@replaced@@TF` can't be used be-
cause the flag is reset at the `\endgroup` above, so we check if `\@curr@file` and
`\@curr@file@reqd` differ. This macro is issued separate from `\set@curr@file` because
it changes `\CurrentFile`, and side-effects would quickly get out of control.

```
374 \def\@filehook@set@CurrentFile{%
375   \@expl@@@filehook@set@curr@file@@nNN{\@curr@file}%
376     \CurrentFileUsed\CurrentFilePathUsed
377   \ifx\@curr@file@reqd\@curr@file
378     \let\CurrentFile\CurrentFileUsed
379     \let\CurrentFilePath\CurrentFilePathUsed
380   \else
381     \@expl@@@filehook@set@curr@file@@nNN{\@curr@file@reqd}%
382       \CurrentFile\CurrentFilePath
383   \fi}
```

(*End definition for* \@filehook@set@CurrentFile , \@kernel@make@file@csname *, and* \@set@curr@file@aux*.
These functions are documented on page* **??**.)

**\@@_set_curr_file:nNN**
**\@@_set_curr_file_assign:nnnNN**

When inputting a file, \set@curr@file does a file lookup (in \input@path and
\l_file_search_path_seq) and returns the actual file name (⟨*base*⟩ plus ⟨*ext*⟩) in
\CurrentFileUsed, and in case there's a file substitution, the requested file in
\CurrentFile (otherwise both are the same). Only the base and extension are returned,
regardless of the input (both path/to/file.tex and file.tex end up as file.tex in
\CurrentFile). The path is returned in \CurrentFilePath, in case it's needed.

```
389 ⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
390 ⟨latexrelease⟩                {@@_set_curr_file:nNN}{Set curr file}%
391 \ExplSyntaxOn
392 ⟨@@=filehook⟩
393 \cs_new_protected:Npn \__filehook_set_curr_file:nNN #1
394   {
395     \exp_args:Nf \__filehook_file_parse_full_name:nN {#1}
396       \__filehook_set_curr_file_assign:nnnNN
397   }
398 \cs_new_protected:Npn \__filehook_set_curr_file_assign:nnnNN #1 #2 #3 #4 #5
399   {
400     \str_set:Nn #5 {#1}
401     \str_set:Nn #4 {#2#3}
402   }
403 \ExplSyntaxOff
404 ⟨/2ekernel | latexrelease⟩
405 ⟨latexrelease⟩\EndIncludeInRelease
406 ⟨*2ekernel⟩
```

(*End definition for* \@@_set_curr_file:nNN *and* \@@_set_curr_file_assign:nnnNN. *These functions are documented on page* **??**.)

## 2.7  Replacing a file and detecting loops

\_\_filehook_resolve_file_subst:w
\_\_filehook_normalize_file_name:w
\_\_filehook_file_name_compose:nnn

Start by sanitizing the file with \__filehook_file_parse_full_name:nN then do \__filehook_file_subst_begin:nnn{⟨path⟩}{⟨name⟩}{⟨ext⟩}.

```
407 ⟨/2ekernel⟩
408 ⟨*2ekernel | latexrelease⟩
409 ⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
410 ⟨latexrelease⟩        {\__filehook_resolve_file_subst:w}{Replace files detect loops}%
411 \ExplSyntaxOn
412 \cs_new:Npn \__filehook_resolve_file_subst:w #1 \@nil
413   { \__filehook_file_parse_full_name:nN {#1} \__filehook_file_subst_begin:nnn }
414 \cs_new:Npn \__filehook_normalize_file_name:w #1 \@nil
415   { \__filehook_file_parse_full_name:nN {#1} \__filehook_file_name_compose:nnn }
416 \cs_new:Npn \__filehook_file_name_compose:nnn #1 #2 #3
417   { \tl_if_empty:nF {#1} { #1 / } #2#3 }
```

flag␣\_\_filehook_file_replaced
\_\_filehook_if_file_replaced:TF
\_\_filehook_clear_replacement_flag:

Since the file replacement is done expandably in a \csname, use a flag to remember if a substitution happened. We use this in \set@curr@file to short-circuit some of it in case no substitution happened (by far the most common case, so it's worth optimizing). The flag raised during the file substitution algorithm must be explicitly cleared after the \__filehook_if_file_replaced:TF conditional is no longer needed, otherwise further uses of \__filehook_if_file_replaced:TF will wrongly return true.

```
418 \flag_new:n { __filehook_file_replaced }
419 \cs_new:Npn \__filehook_if_file_replaced:TF #1 #2
420   { \flag_if_raised:nTF { __filehook_file_replaced } {#1} {#2} }
421 \cs_new_protected:Npn \__filehook_clear_replacement_flag:
422   { \flag_clear:n { __filehook_file_replaced } }
```

\_\_filehook_file_subst_begin:nnn

First off, start by checking if the current file (⟨name⟩ + ⟨ext⟩) has a declared substitution. If not, then just put that as the name (including a possible ⟨path⟩ in this case): this is the default case with no substitutions, so it's the first to be checked. The auxiliary

`\__filehook_file_subst_tortoise_hare:nn` sees that there's no replacement for `#2#3` and does nothing else.

```
423 \cs_new:Npn \__filehook_file_subst_begin:nnn #1 #2 #3
424   {
425     \__filehook_file_subst_tortoise_hare:nn { #2#3 } { #2#3 }
426       { \__filehook_file_name_compose:nnn {#1} {#2} {#3} }
427   }
428 \ExplSyntaxOff
429 ⟨/2ekernel | latexrelease⟩
430 ⟨latexrelease⟩\EndIncludeInRelease
431 ⟨∗2ekernel⟩
```

### 2.7.1 The Tortoise and Hare algorithm

<div style="float:left">

`\__filehook_file_subst_tortoise_hare:nn`
`\__filehook_file_subst_loop:NN`
`\__filehook_file_subst_loop:cc`

</div>

If there is a substitution (⟨*true*⟩ in the first `\cs_if_exist:cTF` below), then first check if there is no substitution down the line: this should be the second most common case, of one file replaced by another. In that case just leave the substitution there and the job is done. If any substitution happens, then the `\flag __filehook_file_replaced` is raised (conditionally, because checking if a flag is raised is much faster than raising it over and over again).

If, however there are more substitutions, then we need to check for a possible loop in the substitutions, which would otherwise put TEX in an infinite loop if just an exhaustive expansion was used.

To detect a loop, the *Tortoise and Hare* algorithm is used. The name of the algorithm is an analogy to Aesop's fable, in which the Hare outruns a Tortoise. The two pointers here are the csnames which contains each file replacement, both of which start at the position zero, which is the file requested. In the inner part of the macro below, `\__filehook_file_subst_loop:cc` is called with `\@file-subst@`⟨*file*⟩ and `\@file-subst@\@file-subst@`⟨*file*⟩; that is, the substitution of ⟨*file*⟩ and the substitution of that substitution: the Tortoise walks one step while the Hare walks two.

Within `\__filehook_file_subst_loop:NN` the two substitutions are compared, and if they lead to the same file it means that there is a loop in the substitutions. If there's no loop, `\__filehook_file_subst_tortoise_hare:nn` is called again with the Tortoise at position 1 and the hare at 2. Again, the substitutions are checked ahead of the Hare pointer to check that it won't run too far; in case there is no loop in the declarations, eventually one of the `\cs_if_exist:cTF` below will go ⟨*false*⟩ and the algorithm will end; otherwise it will run until the Hare reaches the same spot as the tortoise and a loop is detected.

```
432 ⟨/2ekernel⟩
433 ⟨∗2ekernel | latexrelease⟩
434 ⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
435 ⟨latexrelease⟩    {\__filehook_file_subst_tortoise_hare:nn}{Tortoise and Hare}%
436 \ExplSyntaxOn
437 \cs_new:Npn \__filehook_file_subst_tortoise_hare:nn #1 #2 #3
438   {
439     \cs_if_exist:cTF { @file-subst@ #2 }
440       {
441         \flag_if_raised:nF { __filehook_file_replaced }
442           { \flag_raise:n { __filehook_file_replaced } }
443         \cs_if_exist:cTF { @file-subst@ \use:c { @file-subst@ #2 } }
444           {
```

```
445        \__filehook_file_subst_loop:cc
446          { @file-subst@ #1 }
447          { @file-subst@ \use:c { @file-subst@ #2 } }
448        }
449        { \use:c { @file-subst@ #2 } }
450      }
451      { #3 }
452    }
```

This is just an auxiliary to check if a loop was found, and continue the algorithm otherwise. If a loop is found, the `.tex` file is used as fallback and `\__filehook_file_subst_-cycle_error:cN` is called to report the error.

```
453  \cs_new:Npn \__filehook_file_subst_loop:NN #1 #2
454    {
455      \token_if_eq_meaning:NNTF #1 #2
456        {
457          .tex
458          \__filehook_file_subst_cycle_error:cN { @file-subst@ #1 } #1
459        }
460        { \__filehook_file_subst_tortoise_hare:nn {#1} {#2} {#2} }
461    }
462  \cs_generate_variant:Nn \__filehook_file_subst_loop:NN { cc }
```

<span style="float:left">\__filehook_file_subst_cycle_error:NN<br>\__filehook_file_subst_cycle_error:cN</span>
Showing this type of error expandably is tricky, as we have a very limited amount of characters to show and a potentially large list. As a work around, several errors are printed, each showing one step of the loop, until all the error messages combined show the loop.

```
463  \cs_new:Npn \__filehook_file_subst_cycle_error:NN #1 #2
464    {
465      \msg_expandable_error:nnff { latex2e } { file-cycle }
466        {#1} { \use:c { @file-subst@ #1 } }
467      \token_if_eq_meaning:NNF #1 #2
468        { \__filehook_file_subst_cycle_error:cN { @file-subst@ #1 } #2 }
469    }
470  \cs_generate_variant:Nn \__filehook_file_subst_cycle_error:NN { c }
```

And the error message:

```
471  \msg_new:nnn { latex2e } { file-cycle }
472    { File~loop!~#1~replaced~by~#2... }
```

(*End definition for* `\__filehook_resolve_file_subst:w` *and others.*)

```
473  \ExplSyntaxOff
474  ⟨/2ekernel | latexrelease⟩
475  ⟨latexrelease⟩\EndIncludeInRelease
476  ⟨∗2ekernel⟩

477  ⟨@@=⟩
```

## 2.8 Preventing a package from loading

We support the use case of preventing a package from loading but not any other type of files (e.g., classes).

\disable@package@load defines \@pkg-disable@⟨*package*⟩ to expand to some code #2 instead of loading the package.

```
478 ⟨/2ekernel⟩
479 ⟨∗2ekernel | latexrelease⟩
480 ⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
481 ⟨latexrelease⟩                {\disable@package@load}{Disable packages}%
482 \def\disable@package@load#1#2{%
483   \global\@namedef{@pkg-disable@#1.\@pkgextension}{#2}}

484 \def\@disable@packageload@do#1#2{%
485   \@ifundefined{@pkg-disable@#1}{#2}%
486     {\@nameuse{@pkg-disable@#1}}}
```

\reenable@package@load undefines \@pkg-disable@⟨*package*⟩ to reallow loading a package.

```
487 \def\reenable@package@load#1{%
488   \global\expandafter\let
489   \csname @pkg-disable@#1.\@pkgextension \endcsname \@undefined}

490 ⟨/2ekernel | latexrelease⟩
491 ⟨latexrelease⟩\EndIncludeInRelease
492 ⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
493 ⟨latexrelease⟩                {\disable@package@load}{Disable packages}%
494 ⟨latexrelease⟩
495 ⟨latexrelease⟩\let\disable@package@load   \@undefined
496 ⟨latexrelease⟩\let\@disable@packageload@do\@undefined
497 ⟨latexrelease⟩\let\reenable@package@load  \@undefined
498 ⟨latexrelease⟩\EndIncludeInRelease
499 ⟨∗2ekernel⟩
```

(*End definition for* \disable@package@load*,* \reenable@package@load*, and* \@disable@packageload@do*. These functions are documented on page 6.*)

## 2.9  High-level interfaces for LATEX

None so far and the general feeling for now is that the hooks are enough. Packages like filehook, etc., may use them to set up their interfaces (samples are given below) but for the now the kernel will not provide any.

## 2.10  Internal commands needed elsewhere

Here we set up a few horrible (but consistent) LATEX 2ε names to allow for internal commands to be used outside this module (and in parts that still use LATEX 2ε syntax. We have to unset the @@ since we want double "at" sign in place of double underscores.

```
500 ⟨@@=⟩

501 ⟨/2ekernel⟩
502 ⟨∗2ekernel | latexrelease⟩
503 ⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
504 ⟨latexrelease⟩    {\@expl@@@filehook@if@no@extension@@nTF}{2e tmp interfaces}%
505 \ExplSyntaxOn

506 \cs_new_eq:NN \@expl@@@filehook@if@no@extension@@nTF
507                 \__filehook_if_no_extension:nTF
```

```
508 \cs_new_eq:NN \@expl@@@filehook@set@curr@file@@nNN
509                \__filehook_set_curr_file:nNN
510 \cs_new_eq:NN \@expl@@@filehook@resolve@file@subst@@w
511                \__filehook_resolve_file_subst:w
512 \cs_new_eq:NN \@expl@@@filehook@normalize@file@name@@w
513                \__filehook_normalize_file_name:w
514 \cs_new_eq:NN \@expl@@@filehook@if@file@replaced@@TF
515                \__filehook_if_file_replaced:TF
516 \cs_new_eq:NN \@expl@@@filehook@clear@replacement@flag@@
517                \__filehook_clear_replacement_flag:
518 \cs_new_eq:NN \@expl@@@filehook@drop@extension@@N
519                \__filehook_drop_extension:N
520 \cs_new_eq:NN \@expl@@@filehook@file@push@@
521                \__filehook_file_push:
522 \cs_new_eq:NN \@expl@@@filehook@file@pop@@
523                \__filehook_file_pop:
524 \cs_new_eq:NN \@expl@@@filehook@file@pop@assign@@nnnn
525                \__filehook_file_pop_assign:nnnn
526 \ExplSyntaxOff
```

This one specifically has to be undefined because it is left over in the input stream from \InputIfFileExists and executed when latexrelease is loaded. It cannot be \let to \@undefined otherwise it would error as well, so it is \let to \relax to be silently ignored when loading \latexrelease.

```
527 ⟨/2ekernel | latexrelease⟩
528 ⟨latexrelease⟩\EndIncludeInRelease
529 ⟨latexrelease⟩
530 ⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
531 ⟨latexrelease⟩    {\@expl@@@filehook@if@no@extension@@nTF}{2e tmp interfaces}%
532 ⟨latexrelease⟩\let\@expl@@@filehook@file@pop@@\relax
533 ⟨latexrelease⟩\EndIncludeInRelease
534 ⟨∗2ekernel⟩
```

This ends the kernel code in this file.

```
535 ⟨/2ekernel⟩
```

# 3 A sample package for structuring the log output

```
536 ⟨∗structuredlog⟩
537 ⟨@@=filehook⟩
538 \ProvidesExplPackage
539     {structuredlog}{\ltfilehookdate}{\ltfilehookversion}
540     {Structuring the TeX transcript file}
```

\g__filehook_nesting_level_int    Stores the current package nesting level.

```
541 \int_new:N \g__filehook_nesting_level_int
```

Initialise the counter with the number of files in the \@currnamestack (the number of items divided by 3) minus one, because this package is skipped when printing to the log.

```
542 \int_gset:Nn \g__filehook_nesting_level_int
543    { ( \tl_count:N \@currnamestack ) / 3 - 1 }
```

\__filehook_log_file_record:n

This macro is responsible for increasing and decreasing the file nesting level, as well as printing to the log. The argument is either STOPTART or STOP and the action it takes on the nesting integer depends on that.

```
544 \cs_new_protected:Npn \__filehook_log_file_record:n #1
545   {
546     \str_if_eq:nnT {#1} {START} { \int_gincr:N \g__filehook_nesting_level_int }
547     \iow_term:x
548       {
549         \prg_replicate:nn { \g__filehook_nesting_level_int } { = } ~
550         ( LEVEL ~ \int_use:N \g__filehook_nesting_level_int \c_space_tl #1 ) ~
551         \CurrentFileUsed
```

If there was a file replacement, show that as well:

```
552         \str_if_eq:NNF \CurrentFileUsed \CurrentFile
553           { ~ ( \CurrentFile \c_space_tl requested ) }
554         \iow_newline:
555       }
556     \str_if_eq:nnT {#1} {STOP} { \int_gdecr:N \g__filehook_nesting_level_int }
557   }
```

Now just hook the macro above in the generic file/before...

```
558 \AddToHook{file/before}{ \__filehook_log_file_record:n { START } }
```

...and file/after hooks. We don't want to install the file/after hook immediately, because that would mean it is the first time executed when the package finishes. We therefore put the declaration inside \AddToHookNext so that it gets only installed when we have left this package.

```
559 \AddToHookNext{file/after}
560   { \AddToHook{file/after}{ \__filehook_log_file_record:n { STOP } } } }
```

(*End definition for* \__filehook_log_file_record:n.)

```
561 ⟨@@=⟩
562 ⟨/structuredlog⟩
```

# 4   Package emulations

## 4.1   Package **atveryend** emulation

With the new hook management and the hooks in \enddocument all of atveryend is taken care of. We can make an emulation only here after the substitution functionality is available:

```
563 ⟨*2ekernel⟩
564 \declare@file@substitution{atveryend.sty}{atveryend-ltx.sty}
565 ⟨/2ekernel⟩
```

Here is the package file we point to:

```
566 ⟨*atveryend-ltx⟩
567 \ProvidesPackage{atveryend-ltx}
568     [2020/08/19 v1.0a
569       Emulation of the original atveryend package^^Jwith kernel methods]
```

Here are new definitions for its interfaces now pointing to the hooks in `\enddocument`

```
570 \newcommand\AfterLastShipout  {\AddToHook{enddocument/afterlastpage}}
571 \newcommand\AtVeryEndDocument {\AddToHook{enddocument/afteraux}}
```

Next one is a bit of a fake, but the result should normally be as expected. If not, one needs to add a rule to sort the code chunks in `enddocument/info`.

```
572 \newcommand\AtEndAfterFileList{\AddToHook{enddocument/info}}
```

```
573 \newcommand\AtVeryVeryEnd     {\AddToHook{enddocument/end}}
```

`\BeforeClearDocument`  This one is the only one we don't implement or rather don't have a dedicated hook in the code.

```
574 \ExplSyntaxOn
575 \newcommand\BeforeClearDocument[1]
576   { \AtEndDocument{#1}
577     \atveryend@DEPRECATED{BeforeClearDocument \tl_to_str:n{#1}}
578   }
579 \cs_new:Npn\atveryend@DEPRECATED #1
580   {\iow_term:x{======~DEPRECATED~USAGE~#1~=========}}
581 \ExplSyntaxOff
```

(*End definition for* `\BeforeClearDocument`*. This function is documented on page* **??**.)

```
582 ⟨/atveryend-ltx⟩
```

# 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.