The `ltfilehook` documentation[*]

Frank Mittelbach, Phelype Oleinik, LaTeX Project Team

August 22, 2024

Contents

[*]This code has version v1.0o dated 2024/03/13, © 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 `\clearpage`s 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`
`include/.../before`
`include/end`
`include/.../end`
`include/after`
`include/.../after`

These are:

`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]

`include/excluded, include/⟨name⟩/excluded` The above hooks for `\include` files are only executed when the file is loaded (or more exactly the load is attempted). If, however, the `\include` file is explicitly excluded (through an `\includeonly` statement) the above hooks are bypassed and instead the `include/excluded` hook followed by the `include/⟨name⟩/excluded` hook are executed. This happens after LaTeX has loaded the `.aux` file for this include file, i.e., after LaTeX has updated its counters to pretend that the file was seen.

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

If you want to execute code that is run for every `\include` regardless of whether or not it is excluded, use the `cmd/include/before` or `cmd/include/after` 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.

[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.

1.6 Kernel, class, and package interfaces for LaTeX

`\declare@file@substitution`
`\undeclare@file@substitution`

```
\declare@file@substitution   {⟨file⟩} {⟨replacement-file⟩}
\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. As such it is mainly meant for use in the LaTeX kernel but other use cases are conceivable.

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!*

`\disable@package@load`
`\reenable@package@load`

```
\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 and again it should only be applied if there are good reasons for doing this.[3]

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
```

[3] Just to be sure: "I don't like this package by somebody else" is not a good one :-)

```
= (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

```
⟨*2ekernel⟩
⟨@@=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.

```
⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩                 {\CurrentFile}{Hook management file}%
\ExplSyntaxOn
\tl_new:N \CurrentFile
\tl_new:N \CurrentFilePath
\tl_new:N \CurrentFileUsed
\tl_new:N \CurrentFilePathUsed
```

```
\ExplSyntaxOff
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩                 {\CurrentFile}{Hook management file}%
⟨latexrelease⟩
⟨latexrelease⟩\let \CurrentFile         \@undefined
⟨latexrelease⟩\let \CurrentFilePath     \@undefined
⟨latexrelease⟩\let \CurrentFileUsed     \@undefined
⟨latexrelease⟩\let \CurrentFilePathUsed \@undefined
⟨latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

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

2.2 expl3 helpers

```
⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩          {\__filehook_file_parse_full_name:nN}{File helpers}%
\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.

```
\cs_new:Npn \__filehook_file_parse_full_name:nN #1
  {
    \exp_args:Nf \file_parse_full_name_apply:nN
      {
        \exp_args:Nf \__filehook_full_name:nn
          { \file_full_name:n {#1} } {#1}
      }
  }
\cs_new:Npn \__filehook_full_name:nn #1 #2
  {
    \tl_if_empty:nTF {#1}
      { \tl_trim_spaces:n {#2} }
      { \tl_trim_spaces:n {#1} }
  }
```

(*End of 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.

```
\cs_new:Npn \__filehook_if_no_extension:nTF #1
  {
    \exp_args:Ne \tl_if_empty:nTF
      { \file_parse_full_name_apply:nN {#1} \use_iii:nnn }
  }
```

```
\cs_new_protected:Npn \__filehook_drop_extension:N #1
  {
    \tl_gset:Nx #1
      {
        \exp_args:NV \__filehook_file_parse_full_name:nN #1
          \__filehook_drop_extension_aux:nnn
      }
  }
\cs_new:Npn \__filehook_drop_extension_aux:nnn #1 #2 #3
   { \tl_if_empty:nF {#1} { #1 / } #2 }
```

(*End of 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 `\input`s. 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.

```
\tl_new:N \l__filehook_internal_tl
\seq_if_exist:NF \g__filehook_input_file_seq
  { \seq_new:N \g__filehook_input_file_seq }
\cs_new_protected:Npn \__filehook_file_push:
  {
    \seq_gpush:Nx \g__filehook_input_file_seq
      {
        { \CurrentFilePathUsed } { \CurrentFileUsed }
        { \CurrentFilePath     } { \CurrentFile     }
      }
  }
\cs_new_protected:Npn \__filehook_file_pop:
  {
    \seq_gpop:NNTF \g__filehook_input_file_seq \l__filehook_internal_tl
      { \exp_after:wN \__filehook_file_pop_assign:nnnn \l__filehook_internal_tl }
      {
        \msg_error:nnn { latex2e } { should-not-happen }
          { Tried~to~pop~from~an~empty~file~name~stack. }
      }
  }
\cs_new_protected:Npn \__filehook_file_pop_assign:nnnn #1 #2 #3 #4
  {
    \tl_set:Nn \CurrentFilePathUsed {#1}
    \tl_set:Nn \CurrentFileUsed {#2}
    \tl_set:Nn \CurrentFilePath {#3}
    \tl_set:Nn \CurrentFile {#4}
  }
\ExplSyntaxOff
```

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

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

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

```
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩                 {\file_parse_full_name_apply:nN}{Roll forward help}%
⟨latexrelease⟩
⟨latexrelease⟩\ExplSyntaxOn
⟨latexrelease⟩\cs_if_exist:NF\file_parse_full_name_apply:nN
⟨latexrelease⟩{
⟨latexrelease⟩\cs_new:Npn \file_parse_full_name_apply:nN #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \exp_args:Ne \__file_parse_full_name_auxi:nN
⟨latexrelease⟩      { \__kernel_file_name_sanitize:n {#1} }
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_new:Npn \__file_parse_full_name_auxi:nN #1
⟨latexrelease⟩  {
⟨latexrelease⟩    \__file_parse_full_name_area:nw { } #1
⟨latexrelease⟩      / \s__file_stop
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_new:Npn \__file_parse_full_name_area:nw #1 #2 / #3 \s__file_stop
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_empty:nTF {#3}
⟨latexrelease⟩      { \__file_parse_full_name_base:nw { } #2 . \s__file_stop {#1} }
⟨latexrelease⟩      { \__file_parse_full_name_area:nw { #1 / #2 }
⟨latexrelease⟩                                        #3 \s__file_stop }
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_new:Npn \__file_parse_full_name_base:nw #1 #2 . #3 \s__file_stop
⟨latexrelease⟩  {
⟨latexrelease⟩    \tl_if_empty:nTF {#3}
⟨latexrelease⟩      {
⟨latexrelease⟩        \tl_if_empty:nTF {#1}
⟨latexrelease⟩          {
⟨latexrelease⟩            \tl_if_empty:nTF {#2}
⟨latexrelease⟩              { \__file_parse_full_name_tidy:nnnN { } { } }
⟨latexrelease⟩              { \__file_parse_full_name_tidy:nnnN { .#2 } { } }
⟨latexrelease⟩          }
⟨latexrelease⟩          { \__file_parse_full_name_tidy:nnnN {#1} { .#2 } }
⟨latexrelease⟩      }
⟨latexrelease⟩      { \__file_parse_full_name_base:nw { #1 . #2 }
⟨latexrelease⟩                                        #3 \s__file_stop }
⟨latexrelease⟩  }
⟨latexrelease⟩\cs_new:Npn \__file_parse_full_name_tidy:nnnN #1 #2 #3 #4
⟨latexrelease⟩  {
⟨latexrelease⟩    \exp_args:Nee #4
⟨latexrelease⟩      {
⟨latexrelease⟩        \str_if_eq:nnF {#3} { / } { \use_none:n }
⟨latexrelease⟩        #3 \prg_do_nothing:
⟨latexrelease⟩      }
⟨latexrelease⟩      { \use_none:n #1 \prg_do_nothing: }
⟨latexrelease⟩      {#2}
⟨latexrelease⟩  }
⟨latexrelease⟩}
⟨latexrelease⟩\ExplSyntaxOff
⟨latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩

⟨@@=⟩
```

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 2ε core commands, because of some circular dependencies in the kernel we do this only now and not in `ltfiles`.

`\InputIfFileExists`
`\@input@file@exists@with@hooks`
`\unqu@tefilef@und`

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

```
⟨/2ekernel⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩          {\InputIfFileExists}{Hook management (files)}%
⟨*2ekernel | latexrelease⟩
```

```
\let\InputIfFileExists\@undefined
\DeclareRobustCommand \InputIfFileExists[2]{%
  \IfFileExists{#1}%
    {%
      \@expl@@@filehook@file@push@@
      \@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.

```
      \expandafter\@swaptwoargs\expandafter
        {\expandafter\@input@file@exists@with@hooks
          \expandafter{\@filef@und}}%
        {#2}%
      \@expl@@@filehook@file@pop@@
    }%
}
\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`.

```
  \edef\reserved@a{%
    \@expl@@@filehook@file@pop@assign@@nnnn
      {\CurrentFilePathUsed}%
      {\CurrentFileUsed}%
      {\CurrentFilePath}%
      {\CurrentFile}}%
  \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 `\detokenized`. 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 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).

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

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

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

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

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

Now declare the non-generic file hooks used above:

```
\NewHook{file/before}
\NewReversedHook{file/after}
⟨latexrelease⟩\EndIncludeInRelease
⟨/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`'.

```
⟨latexrelease⟩\IncludeInRelease{2019/10/01}%
⟨latexrelease⟩             {\InputIfFileExists}{Hook management (files)}%
⟨latexrelease⟩
⟨latexrelease⟩\DeclareRobustCommand \InputIfFileExists[2]{%
⟨latexrelease⟩  \IfFileExists{#1}%
⟨latexrelease⟩    {%
⟨latexrelease⟩  \expandafter\@swaptwoargs\expandafter
⟨latexrelease⟩      {\@filef@und}{#2\@addtofilelist{#1}\@@input}}}
⟨latexrelease⟩\let\@input@file@exists@with@hooks\@undefined
⟨latexrelease⟩\let\unqu@tefilef@und\@undefined
⟨latexrelease⟩\EndIncludeInRelease
```

```
⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩           {\InputIfFileExists}{Hook management (files)}%
⟨latexrelease⟩\long\def \InputIfFileExists#1#2{%
⟨latexrelease⟩  \IfFileExists{#1}%
⟨latexrelease⟩    {#2\@addtofilelist{#1}\@@input \@filef@und}}
```

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

```
⟨latexrelease⟩\expandafter\let\csname InputIfFileExists \endcsname\@undefined
```

```
⟨latexrelease⟩\let\@input@file@exists@with@hooks\@undefined
⟨latexrelease⟩\let\unqu@tefilef@und\@undefined
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

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

2.5 Declaring a file substitution

```
⟨@@=filehook⟩
```

```
⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩           {\__filehook_subst_add:nn}{Declaring file substitution}%
\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).

```
\cs_new_protected:Npn \__filehook_subst_add:nn #1 #2
  {
    \group_begin:
      \cs_set:cpx { } { \exp_not:o { \cs:w\cs_end: } }
      \int_set:Nn \tex_escapechar:D { -1 }
      \cs_gset:cpx
        {
          @file-subst@
          \__filehook_subst_file_normalize:Nn \use_ii_iii:nnn {#1}
        }
        { \__filehook_subst_file_normalize:Nn \__filehook_file_name_compose:nnn
                                    {#2} }
    \group_end:
  }
\cs_new_protected:Npn \__filehook_subst_remove:n #1
  {
    \group_begin:
      \cs_set:cpx { } { \exp_not:o { \cs:w\cs_end: } }
      \int_set:Nn \tex_escapechar:D { -1 }
      \cs_undefine:c
        {
          @file-subst@
          \__filehook_subst_file_normalize:Nn \use_ii_iii:nnn {#1}
        }
    \group_end:
  }
\cs_new:Npn \__filehook_subst_file_normalize:Nn #1 #2
  {
    \exp_after:wN \__filehook_subst_empty_name_chk:NN
      \cs:w \exp_after:wN \cs_end:
        \cs:w \__filehook_file_parse_full_name:nN {#2} #1 \cs_end:
  }
\cs_new:Npn \__filehook_subst_empty_name_chk:NN #1 #2
  { \if_meaning:w #1 #2 .tex \else: \token_to_str:N #2 \fi: }
```

(*End of 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

```
\cs_gset:Npn \use_ii_iii:nnn #1 #2 #3 {#2 #3}
```

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

```
\ExplSyntaxOff
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

`\declare@file@substitution` `\undeclare@file@substitution` 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).

```
⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩          {\declare@file@substitution}{File substitution}%
\ExplSyntaxOn
\cs_new_eq:NN \declare@file@substitution   \__filehook_subst_add:nn
\cs_new_eq:NN \undeclare@file@substitution \__filehook_subst_remove:n
\ExplSyntaxOff
⟨/2ekernel | latexrelease⟩
⟨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.

```
⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩          {\declare@file@substitution}{File substitution}%
⟨latexrelease⟩
⟨latexrelease⟩\let \declare@file@substitution   \@gobbletwo
⟨latexrelease⟩\let \undeclare@file@substitution \@gobble
⟨latexrelease⟩
```

```
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

(*End of definition for* `\declare@file@substitution` *and* `\undeclare@file@substitution`*. These functions are documented on page 6.*)

```
⟨@@=⟩
```

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.

```
⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2022/06/01}%
⟨latexrelease⟩          {\set@curr@file}{Setting current file name}%
\def\set@curr@file{%
  \begingroup
    \set@curr@file@aux}
\edef\set@curr@file@nosearch{%
  \begingroup
    \let\noexpand\input@path\noexpand\@empty
    \csname seq_clear:N\endcsname
      \expandafter\noexpand\csname l_file_search_path_seq\endcsname
    \noexpand\set@curr@file@aux}
\def\set@curr@file@aux#1{%
    \escapechar\m@ne
    \let\protect\string
    \edef~{\string~}%
    \expandafter\def\csname\expandafter\endcsname
      \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.

```
    \@expl@@@filehook@if@no@extension@@nTF{#1}%
      {\@tempswatrue}{\@tempswafalse}%
    \@kernel@make@file@csname\@curr@file
      \@expl@@@filehook@resolve@file@subst@@w {#1}%
    \@expl@@@filehook@if@file@replaced@@TF
      {\@kernel@make@file@csname\@curr@file@reqd
         \@expl@@@filehook@normalize@file@name@@w{#1}%
       \if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file@reqd \fi}%
      {\if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file \fi
       \global\let\@curr@file@reqd\@curr@file}%
    \@expl@@@filehook@clear@replacement@flag@@
  \endgroup}
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
```

```
⟨latexrelease⟩\IncludeInRelease{2021/06/01}%
⟨latexrelease⟩          {\set@curr@file}{Setting current file name}%
⟨latexrelease⟩\def\set@curr@file#1{%
⟨latexrelease⟩  \begingroup
⟨latexrelease⟩    \escapechar\m@ne
⟨latexrelease⟩    \let\protect\string
⟨latexrelease⟩    \edef~{\string~}%
⟨latexrelease⟩    \expandafter\def\csname\expandafter\endcsname
⟨latexrelease⟩      \expandafter{\csname\endcsname}%
⟨latexrelease⟩    \@expl@@@filehook@if@no@extension@@nTF{#1}%
⟨latexrelease⟩      {\@tempswatrue}{\@tempswafalse}%
⟨latexrelease⟩    \@kernel@make@file@csname\@curr@file
⟨latexrelease⟩      \@expl@@@filehook@resolve@file@subst@@w {#1}%
⟨latexrelease⟩    \@expl@@@filehook@if@file@replaced@@TF
⟨latexrelease⟩      {\@kernel@make@file@csname\@curr@file@reqd
⟨latexrelease⟩         \@expl@@@filehook@normalize@file@name@@w{#1}%
⟨latexrelease⟩       \if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file@reqd \fi}%
⟨latexrelease⟩      {\if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file \fi
⟨latexrelease⟩       \global\let\@curr@file@reqd\@curr@file}%
⟨latexrelease⟩    \@expl@@@filehook@clear@replacement@flag@@
⟨latexrelease⟩  \endgroup}
⟨latexrelease⟩\let\set@curr@file@nosearch\@undefined
⟨latexrelease⟩\EndIncludeInRelease
```

```
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩          {\set@curr@file}{Setting current file name}%
⟨latexrelease⟩\def\set@curr@file#1{%
⟨latexrelease⟩  \begingroup
⟨latexrelease⟩    \escapechar\m@ne
⟨latexrelease⟩    \expandafter\def\csname\expandafter\endcsname
⟨latexrelease⟩      \expandafter{\csname\endcsname}%
⟨latexrelease⟩    \@expl@@@filehook@if@no@extension@@nTF{#1}%
⟨latexrelease⟩      {\@tempswatrue}{\@tempswafalse}%
⟨latexrelease⟩    \@kernel@make@file@csname\@curr@file
⟨latexrelease⟩      \@expl@@@filehook@resolve@file@subst@@w {#1}%
⟨latexrelease⟩    \@expl@@@filehook@if@file@replaced@@TF
⟨latexrelease⟩      {\@kernel@make@file@csname\@curr@file@reqd
⟨latexrelease⟩         \@expl@@@filehook@normalize@file@name@@w{#1}%
⟨latexrelease⟩       \if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file@reqd \fi}%
⟨latexrelease⟩      {\if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file \fi
⟨latexrelease⟩       \global\let\@curr@file@reqd\@curr@file}%
⟨latexrelease⟩    \@expl@@@filehook@clear@replacement@flag@@
⟨latexrelease⟩  \endgroup}
⟨latexrelease⟩\let\set@curr@file@nosearch\@undefined
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{2019/10/01}%
⟨latexrelease⟩          {\set@curr@file}{Setting current file name}%
⟨latexrelease⟩\def\set@curr@file#1{%
⟨latexrelease⟩  \begingroup
⟨latexrelease⟩    \escapechar\m@ne
⟨latexrelease⟩    \xdef\@curr@file{%
⟨latexrelease⟩      \expandafter\expandafter\expandafter\unquote@name
⟨latexrelease⟩      \expandafter\expandafter\expandafter{%
⟨latexrelease⟩      \expandafter\string
⟨latexrelease⟩        \csname\@firstofone#1\@empty\endcsname}}%
⟨latexrelease⟩  \endgroup
⟨latexrelease⟩}
⟨latexrelease⟩\let\set@curr@file@nosearch\@undefined
⟨latexrelease⟩\EndIncludeInRelease

⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩          {\set@curr@file}{Setting current file name}%
⟨latexrelease⟩\let\set@curr@file\@undefined
⟨latexrelease⟩\let\set@curr@file@nosearch\@undefined
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

(*End of 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*

```
⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩          {\@kernel@make@file@csname}{Make file csname}%

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

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

```
\long\def\@set@curr@file@aux#1{%
  \expandafter\ifx\csname\endcsname#1%
    .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 because 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.

```
\def\@filehook@set@CurrentFile{%
```

```
  \@expl@@@filehook@set@curr@file@@nNN{\@curr@file}%
    \CurrentFileUsed\CurrentFilePathUsed
  \ifx\@curr@file@reqd\@curr@file
    \let\CurrentFile\CurrentFileUsed
    \let\CurrentFilePath\CurrentFilePathUsed
  \else
    \@expl@@@filehook@set@curr@file@@nNN{\@curr@file@reqd}%
      \CurrentFile\CurrentFilePath
  \fi}
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

(*End of 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.

```
⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩            {@@_set_curr_file:nNN}{Set curr file}%
\ExplSyntaxOn
⟨@@=filehook⟩
\cs_new_protected:Npn \__filehook_set_curr_file:nNN #1
  {
    \exp_args:Nf \__filehook_file_parse_full_name:nN {#1}
      \__filehook_set_curr_file_assign:nnnNN
  }
\cs_new_protected:Npn \__filehook_set_curr_file_assign:nnnNN #1 #2 #3 #4 #5
  {
    \str_set:Nn #5 {#1}
    \str_set:Nn #4 {#2#3}
  }
\ExplSyntaxOff
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

(*End of 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*⟩}.

```
⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩      {\__filehook_resolve_file_subst:w}{Replace files detect loops}%
```

```
\ExplSyntaxOn
\cs_new:Npn \__filehook_resolve_file_subst:w #1 \@nil
  { \__filehook_file_parse_full_name:nN {#1} \__filehook_file_subst_begin:nnn }
\cs_new:Npn \__filehook_normalize_file_name:w #1 \@nil
  { \__filehook_file_parse_full_name:nN {#1} \__filehook_file_name_compose:nnn }
\cs_new:Npn \__filehook_file_name_compose:nnn #1 #2 #3
  { \tl_if_empty:nF {#1} { #1 / } #2#3 }
```

`__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.

```
\flag_new:n { __filehook_file_replaced }
\cs_new:Npn \__filehook_if_file_replaced:TF #1 #2
  { \flag_if_raised:nTF { __filehook_file_replaced } {#1} {#2} }
\cs_new_protected:Npn \__filehook_clear_replacement_flag:
  { \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.

```
\cs_new:Npn \__filehook_file_subst_begin:nnn #1 #2 #3
  {
    \__filehook_file_subst_tortoise_hare:nn { #2#3 } { #2#3 }
      { \__filehook_file_name_compose:nnn {#1} {#2} {#3} }
  }
\ExplSyntaxOff
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

2.7.1 The Tortoise and Hare algorithm

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

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.

```
⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩  {\__filehook_file_subst_tortoise_hare:nn}{Tortoise and Hare}%
\ExplSyntaxOn
\cs_new:Npn \__filehook_file_subst_tortoise_hare:nn #1 #2 #3
  {
    \cs_if_exist:cTF { @file-subst@ #2 }
      {
        \flag_if_raised:nF { __filehook_file_replaced }
          { \flag_raise:n { __filehook_file_replaced } }
        \cs_if_exist:cTF { @file-subst@ \use:c { @file-subst@ #2 } }
          {
            \__filehook_file_subst_loop:cc
              { @file-subst@ #1 }
              { @file-subst@ \use:c { @file-subst@ #2 } }
          }
          { \use:c { @file-subst@ #2 } }
      }
      { #3 }
  }
```

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.

```
\cs_new:Npn \__filehook_file_subst_loop:NN #1 #2
  {
    \token_if_eq_meaning:NNTF #1 #2
      {
        .tex
        \__filehook_file_subst_cycle_error:cN { @file-subst@ #1 } #1
      }
      { \__filehook_file_subst_tortoise_hare:nn {#1} {#2} {#2} }
  }
\cs_generate_variant:Nn \__filehook_file_subst_loop:NN { cc }
```

`\__filehook_file_subst_cycle_error:NN`
`\__filehook_file_subst_cycle_error:cN`

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.

```
\cs_new:Npn \__filehook_file_subst_cycle_error:NN #1 #2
  {
    \msg_expandable_error:nnff { latex2e } { file-cycle }
```

```
      {#1} { \use:c { @file-subst@ #1 } }
    \token_if_eq_meaning:NNF #1 #2
      { \__filehook_file_subst_cycle_error:cN { @file-subst@ #1 } #2 }
  }
\cs_generate_variant:Nn \__filehook_file_subst_cycle_error:NN { c }
```

And the error message:

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

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

```
\ExplSyntaxOff
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
⟨@@=⟩
```

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`
`\reenable@package@load`
`\@disable@packageload@do`

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

```
⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩            {\disable@package@load}{Disable packages}%
\def\disable@package@load#1#2{%
  \global\@namedef{@pkg-disable@#1.\@pkgextension}{#2}}
```

Here we check if a control sequence named `\@pkg-disable@`⟨*name*⟩`.sty` is defined, and if so don't use the package loading code `#2`, but use the replacement code stored in that control sequence, write something to the log, and then prevent `\@onefilewithoptions` from sanity-checking the requested package date (the `\expandafter` here triggers one in `\@onefilewithoptions` that ends a conditional there, and the `\@gobbletwo` removes the date checking code from the input stream).

```
\def\@disable@packageload@do#1#2{%
  \@ifundefined{@pkg-disable@#1}%
     {#2}%
     {\@nameuse{@pkg-disable@#1}%
      \@latex@info{Package '#1' has been disabled.%
        \MessageBreak Load request ignored}%
      \expandafter\@gobbletwo}}
```

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

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

```
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩          {\disable@package@load}{Disable packages}%
⟨latexrelease⟩
⟨latexrelease⟩\let\disable@package@load   \@undefined
⟨latexrelease⟩\let\@disable@packageload@do\@undefined
⟨latexrelease⟩\let\reenable@package@load  \@undefined
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

(*End of 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 2_ε names to allow for internal commands to be used outside this module (and in parts that still use LaTeX 2_ε syntax. We have to unset the `@@` since we want double "at" sign in place of double underscores.

```
⟨@@=⟩

⟨/2ekernel⟩
⟨*2ekernel | latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{2020/10/01}%
⟨latexrelease⟩    {\@expl@@@filehook@if@no@extension@@nTF}{2e tmp interfaces}%
\ExplSyntaxOn

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

\cs_new_eq:NN \@expl@@@filehook@set@curr@file@@nNN
              \__filehook_set_curr_file:nNN

\cs_new_eq:NN \@expl@@@filehook@resolve@file@subst@@w
              \__filehook_resolve_file_subst:w

\cs_new_eq:NN \@expl@@@filehook@normalize@file@name@@w
              \__filehook_normalize_file_name:w

\cs_new_eq:NN \@expl@@@filehook@if@file@replaced@@TF
              \__filehook_if_file_replaced:TF

\cs_new_eq:NN \@expl@@@filehook@clear@replacement@flag@@
              \__filehook_clear_replacement_flag:

\cs_new_eq:NN \@expl@@@filehook@drop@extension@@N
              \__filehook_drop_extension:N

\cs_new_eq:NN \@expl@@@filehook@file@push@@
              \__filehook_file_push:

\cs_new_eq:NN \@expl@@@filehook@file@pop@@
              \__filehook_file_pop:
```

```
\cs_new_eq:NN \@expl@@@filehook@file@pop@assign@@nnnn
              \__filehook_file_pop_assign:nnnn
```

```
\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`.

```
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩
⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩    {\@expl@@@filehook@if@no@extension@@nTF}{2e tmp interfaces}%
⟨latexrelease⟩\let\@expl@@@filehook@file@pop@@\relax
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

This ends the kernel code in this file.

```
⟨/2ekernel⟩
```

3 A sample package for structuring the log output

```
⟨*structuredlog⟩
⟨@@=filehook⟩
```

```
\ProvidesExplPackage
    {structuredlog}{\ltfilehookdate}{\ltfilehookversion}
    {Structuring the TeX transcript file}
```

\g__filehook_nesting_level_int

Stores the current package nesting level.

```
\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.

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

(*End of definition for* `\g__filehook_nesting_level_int`*.*)

__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.

```
\cs_new_protected:Npn \__filehook_log_file_record:n #1
  {
    \str_if_eq:nnT {#1} {START} { \int_gincr:N \g__filehook_nesting_level_int }
    \iow_term:x
      {
        \prg_replicate:nn { \g__filehook_nesting_level_int } { = } ~
        ( LEVEL ~ \int_use:N \g__filehook_nesting_level_int \c_space_tl #1 ) ~
        \CurrentFileUsed
```

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

```
        \str_if_eq:NNF \CurrentFileUsed \CurrentFile
          { ~ ( \CurrentFile \c_space_tl requested ) }
        \iow_newline:
      }
    \str_if_eq:nnT {#1} {STOP} { \int_gdecr:N \g__filehook_nesting_level_int }
  }
```

Now just hook the macro above in the generic `file/before`…

```
\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.

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

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

```
⟨@@=⟩
⟨/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:

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

Here is the package file we point to:

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

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

```
\newcommand\AfterLastShipout  {\AddToHook{enddocument/afterlastpage}}
\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`.

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

```
\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.

```
\ExplSyntaxOn
\newcommand\BeforeClearDocument[1]
  { \AtEndDocument{#1}
    \atveryend@DEPRECATED{BeforeClearDocument \tl_to_str:n{#1}}
  }
```

```
\cs_new:Npn\atveryend@DEPRECATED #1
  {\iow_term:x{======~DEPRECATED~USAGE~#1~==========}}
\ExplSyntaxOff
```

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

```
⟨/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.