

The `ltfilehook` package*

Frank Mittelbach

October 1, 2020

Contents

1	Introduction	2
1.1	Provided hooks	2
1.2	General hooks for file reading	2
1.3	Hooks for package and class files	3
1.4	Hooks for <code>\include</code> files	4
1.5	High-level interfaces for <code>L^AT_EX</code>	5
1.6	Internal interfaces for <code>L^AT_EX</code>	5
1.7	A sample package for structuring the log output	5
2	The Implementation	6
2.1	Document and package-level commands	6
2.2	<code>expl3</code> helpers	7
2.3	Declaring the file-related hooks	9
2.4	Patching <code>L^AT_EX</code> 's <code>\InputIfFileExists</code> command	9
2.5	Declaring a file substitution	10
2.6	Selecting a file (<code>\set@curr@file</code>)	12
2.7	Replacing a file and detecting loops	14
2.7.1	The Tortoise and Hare algorithm	15
2.8	Preventing a package from loading	17
2.9	High-level interfaces for <code>L^AT_EX</code>	17
2.10	Internal commands needed elsewhere	18
3	A sample package for structuring the log output	18
4	Package emulations	19
4.1	Package <code>atveryend</code> emulation	19
	Index	20

*This package has version v1.0b dated 2020/09/26, © `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`.

<code>file/before</code>
<code>file/before/...</code>
<code>file/after/...</code>
<code>file/after</code>

These are:

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

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

```
\UseHook{⟨file/before⟩}  
\UseHook{⟨file/before/⟨file name⟩⟩}  
  ⟨file contents⟩  
\UseHook{⟨file/after/⟨file name⟩⟩}  
\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 T_EX prioritizes `.tex` files, so if two files `foo` and `foo.tex` exist in the search path, only the latter will be seen.

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

\CurrentFile

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

\CurrentFilePath

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

\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/after/array.sty` would be called after the `array` package got loaded. But as packages and classes form as special group of files, there are some additional hooks available that only apply when a package or class is loaded.

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

These are:

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

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

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

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

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

```
\UseHook{<package/before>}
\UseHook{<package/before/<package name>>}}
\UseHook{<file/before>}
```

```

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

```

code from \AtEndOfPackage if used inside the package

```

\UseHook{\package/after/\package name}}
\UseHook{\package/after}}

```

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

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

1.4 Hooks for `\include` files

To manage `\include` files, L^AT_EX 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 L^AT_EX makes an attempt to load the `\include` file (even if it doesn't exist and all that happens is “No file `\filename.tex`”).

```

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

```

These are:

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

include/after/`\name`, include/after These hooks are executed (in that order) after L^AT_EX has issued the `\clearpage` but before it 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

¹If 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`.

²For 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.

1.5 High-level interfaces for L^AT_EX

We do not provide any high-level L^AT_EX commands (like `filehook` or `scrfile` do) but think that for package writers the commands from for hook management are sufficient.

1.6 Internal interfaces for L^AT_EX

<code>\declare@file@substitution</code>	<code>\declare@file@substitution {<file>} {<replacement-file>}</code>
<code>\undecclare@file@substitution</code>	<code>\undecclare@file@substitution {<file>}</code>

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 L^AT_EX 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 `\undecclare@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 implementating the same functionality as the original one!

<code>\disable@package@load</code>	<code>\disable@package@load {<package>} {<alternate-code>}</code>
<code>\reenable@package@load</code>	<code>\reenable@package@load {<package>}</code>

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) om1lmm.fd
== (LEVEL 2 STOP) om1lmm.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` 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 L^AT_EX, 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
\_filehook_set_curr_file_assign:nnnNN

```

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 normalisation 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`, `_filehook_full_name:nn`, and `_filehook_set_curr_file_assign:nnnNN`.)

```

\_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_new:N \g__filehook_input_file_seq
61 \cs_new_protected:Npn \__filehook_file_push:
62   {
63     \seq_gpush:Nx \g__filehook_input_file_seq
64     {
65       { \CurrentFilePathUsed } { \CurrentFileUsed }
66       { \CurrentFilePath      } { \CurrentFile      }
67     }
68   }
69 \cs_new_protected:Npn \__filehook_file_pop:
70   {
71     \seq_gpop:NNTF \g__filehook_input_file_seq \l__filehook_internal_tl
72     { \exp_after:wN \__filehook_file_pop_assign:nnnn \l__filehook_internal_tl }
73     {
74       \msg_error:nnn { hooks } { should-not-happen }
75       { Tried~to~pop~from~an~empty~file~name~stack. }
76     }
77   }
78 \cs_new_protected:Npn \__filehook_file_pop_assign:nnnn #1 #2 #3 #4
79   {
80     \tl_set:Nn \CurrentFilePathUsed {#1}
81     \tl_set:Nn \CurrentFileUsed {#2}
82     \tl_set:Nn \CurrentFilePath {#3}
83     \tl_set:Nn \CurrentFile {#4}
84   }
85 \ExplSyntaxOff

```

(End definition for \g__filehook_input_file_seq and others.)

```

86 </2ekernel | latexrelease>
87 <latexrelease>\EndIncludeInRelease
88 <*2ekernel>
89 <@@=>

```


2.3 Declaring the file-related hooks

All hooks starting with `file/`, `include/`, `class/` or `package/` are generic and will be allocated if code is added to them. Thus there is no need to explicitly declare any hook in the code below.

Furthermore, those named `.../after` or `.../end` are automatically declared as reversed hooks if filled with code, so this is also automatically taken care of.

2.4 Patching L^AT_EX’s `\InputIfFileExists` command

Most of what we have to do is adding `\UseHook` into several L^AT_EX 2_ε core commands, because of some circular dependencies in the kernel we do this only now and not in `ltxfiles`.

`\InputIfFileExists` `\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.

```

90 </2ekernel>
91 <latexrelease>\IncludeInRelease{2020/10/01}%
92 <latexrelease>          {\InputIfFileExists}{Hook management (files)}%
93 <*2ekernel|latexrelease>
94 \let\InputIfFileExists\@undefined
95 \DeclareRobustCommand \InputIfFileExists[2]{%
96   \IfFileExists{#1}%
97   {%
98     \@expl@@@filehook@file@push@@
99     \@filehook@set@CurrentFile

```

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 restored 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`.

```

100   \edef\reserved@a{\@filef@und
101     \@expl@@@filehook@file@pop@assign@@nnnn
102     {\CurrentFilePathUsed}%
103     {\CurrentFileUsed}%
104     {\CurrentFilePath}%
105     {\CurrentFile}}%
106   \expandafter\@swaptwoargs\expandafter
107   {\reserved@a}%
108   {%
109     #2%
110     \@addtofilelist{#1}%
111     \UseHook{file/before}%

```

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

```
112 \UseHook{file/before/\CurrentFile}%
113 \@@input
114 }%
```

And it is restored here so we can use it once more.

```
115 \UseHook{file/after/\CurrentFile}%
116 \UseHook{file/after}%
117 \@expl@@filehook@file@pop@@
118 }%
119 }
120 \<latexrelease>\EndIncludeInRelease
121 \</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’.

```
122 \<latexrelease>\IncludeInRelease{2019/10/01}%
123 \<latexrelease> \InputIfFileExists{Hook management (files)}%
124 \<latexrelease>
125 \<latexrelease>\DeclareRobustCommand \InputIfFileExists[2]{%
126 \<latexrelease> \IfFileExists{#1}%
127 \<latexrelease> {%
128 \<latexrelease> \expandafter\@swaptwoargs\expandafter
129 \<latexrelease> {\@filef@und}{#2\@addtofilelist{#1}\@@input}}%
130 \<latexrelease>\EndIncludeInRelease
131 \<latexrelease>\IncludeInRelease{0000/00/00}%
132 \<latexrelease> \InputIfFileExists{Hook management (files)}%
133 \<latexrelease>\long\def \InputIfFileExists#1#2{%
134 \<latexrelease> \IfFileExists{#1}%
135 \<latexrelease> {#2\@addtofilelist{#1}\@@input \@filef@und}}
136 \<latexrelease>\EndIncludeInRelease
137 \<*2ekernel>
```

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

2.5 Declaring a file substitution

```
138 \<@@=filehook>
139 \</2ekernel>
140 \<*2ekernel|latexrelease>
141 \<latexrelease>\IncludeInRelease{2020/10/01}%
142 \<latexrelease> \_filehook_subst_add:nn{Declaring file substitution}%
143 \ExplSyntaxOn
```

`_filehook_subst_add:nn` `_filehook_substitution_lthooadd: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 csnam is used to check that).

```
144 \cs_new_protected:Npn \_filehook_subst_add:nn #1 #2
145 {
146 \group_begin:
147 \cs_set:cpx { } { \exp_not:o { \cs:w\cs_end: } }
```

```

148 \int_set:Nn \tex_escapechar:D { -1 }
149 \cs_gset:cpx { @file-subst@ \_filehook_subst_file_normalize:n {#1} }
150 { \_filehook_subst_file_normalize:n {#2} }
151 \group_end:
152 }
153 \cs_new_protected:Npn \_filehook_subst_remove:n #1
154 {
155 \group_begin:
156 \cs_set:cpx { } { \exp_not:o { \cs:w\cs_end: } }
157 \int_set:Nn \tex_escapechar:D { -1 }
158 \cs_undefine:c { @file-subst@ \_filehook_subst_file_normalize:n {#1} }
159 \group_end:
160 }
161 \cs_new:Npn \_filehook_subst_file_normalize:n #1
162 {
163 \exp_after:wN \_filehook_subst_empty_name_chk:NN
164 \cs:w \exp_after:wN \cs_end:
165 \cs:w \_filehook_file_parse_full_name:nN {#1} \use_ii_iii:nnn \cs_end:
166 }
167 \cs_new:Npn \_filehook_subst_empty_name_chk:NN #1 #2
168 { \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

```

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

```

(End definition for \use_ii_iii:nnn.)

```

170 \ExplSyntaxOff
171 </2ekernel | latexrelease>
172 <latexrelease>\EndIncludeInRelease
173 <*2ekernel>

```

\declare@file@substitution For two internals we provide L^AT_EX 2_ε 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., `scrfile`).

\undecclare@file@substitution

```

174 </2ekernel>
175 <*2ekernel | latexrelease>
176 <latexrelease>\IncludeInRelease{2020/10/01}%
177 <latexrelease> \{ \declare@file@substitution \} {File substitution}%
178 \ExplSyntaxOn
179 \cs_new_eq:NN \declare@file@substitution \_filehook_subst_add:nn
180 \cs_new_eq:NN \undecclare@file@substitution \_filehook_subst_remove:n
181 \ExplSyntaxOff
182 </2ekernel | latexrelease>
183 <latexrelease>\EndIncludeInRelease
184 <latexrelease>\IncludeInRelease{0000/00/00}%
185 <latexrelease> \{ \declare@file@substitution \} {File substitution}%
186 <latexrelease>
187 <latexrelease>\let \declare@file@substitution \@undefined
188 <latexrelease>\let \undecclare@file@substitution \@undefined
189 <latexrelease>
190 <latexrelease>\EndIncludeInRelease
191 <*2ekernel>

```

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

```
192 <@@=>
193 \ExplSyntaxOff
```

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

`\set@curr@file` Now we hook into `\set@curr@file` to resolve a possible file substitution, and add `\@curr@file` `\@expl@@@filehook@set@curr@file@@N` at the end, after `\@curr@file` is set.
`\@curr@file@reqd` A file name is built using `\expandafter\string\csname{filename}\endcsname` to avoid expanding utf8 active characters. The `\csname` expands the normalisation 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.

```
194 </2kernel>
195 <*2kernel | latexrelease>
196 <latexrelease>\IncludeInRelease{2020/10/01}%
197 <latexrelease>{\set@curr@file}{Setting current file name}%
198 \def\set@curr@file#1{%
199   \begingroup
200     \escapechar\m@ne
201     \expandafter\def\csname\expandafter\endcsname
202     \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 normalisation 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.

```
203   \@expl@@@filehook@if@no@extension@@nTF{#1}%
204   {\@tempwattrue}{\@tempwafalse}%
205   \@kernel@make@file@csname\@curr@file
206   \@expl@@@filehook@resolve@file@subst@@w {#1}%
207   \@expl@@@filehook@if@file@replaced@@TF
208   {\@kernel@make@file@csname\@curr@file@reqd
209     \@expl@@@filehook@normalize@file@name@@w{#1}%
210     \if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file@reqd \fi}%
211   {\if@tempswa \@expl@@@filehook@drop@extension@@N\@curr@file \fi
212     \global\let\@curr@file@reqd\@curr@file}%
213 \endgroup}
```

```

214 </2ekernel | latexrelease>
215 <latexrelease>\EndIncludeInRelease

216 <latexrelease>\IncludeInRelease{2019/10/01}%
217 <latexrelease>          {\set@curr@file}{Setting current file name}%
218 <latexrelease>\def\set@curr@file#1{%
219 <latexrelease>  \begingroup
220 <latexrelease>    \escapechar\m@ne
221 <latexrelease>    \xdef\@curr@file{%
222 <latexrelease>      \expandafter\expandafter\expandafter\unquote@name
223 <latexrelease>      \expandafter\expandafter\expandafter{%
224 <latexrelease>      \expandafter\string
225 <latexrelease>      \csname\@firstofone#1\@empty\endcsname}}%
226 <latexrelease>  \endgroup
227 <latexrelease>}
228 <latexrelease>\EndIncludeInRelease

229 <latexrelease>\IncludeInRelease{0000/00/00}%
230 <latexrelease>          {\set@curr@file}{Setting current file name}%
231 <latexrelease>\let\set@curr@file\@undefined
232 <latexrelease>\EndIncludeInRelease
233 <*2ekernel>

```

(End definition for \set@curr@file, \@curr@file, and \@curr@file@reqd. 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

```

234 </2ekernel>
235 <*2ekernel | latexrelease>
236 <latexrelease>\IncludeInRelease{2020/10/01}%
237 <latexrelease>          {\@kernel@make@file@csname}{Make file csname}%

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

```

This auxiliary compares \<filename> with \csname\endcsname to check if the empty .tex file was requested.

```

241 \def\@set@curr@file@aux#1{%
242   \expandafter\ifx\csname\endcsname#1%
243     .tex\else\string#1\fi}

```

Then we call \@expl@@@filehook@set@curr@file@@N 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@ 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.

```

244 \def\@filehook@set@CurrentFile{%
245   \@expl@@@filehook@set@curr@file@@N{\@curr@file}%
246   \CurrentFileUsed\CurrentFilePathUsed
247   \ifx\@curr@file@reqd\@curr@file
248     \let\CurrentFile\CurrentFileUsed
249     \let\CurrentFilePath\CurrentFilePathUsed
250   \else
251     \@expl@@@filehook@set@curr@file@@N{\@curr@file@reqd}%

```

```

252 \CurrentFile\CurrentFilePath
253 \fi}
254 </2ekernel | latexrelease>
255 <latexrelease>\EndIncludeInRelease
256 <*2ekernel>

```

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

\@@_set_curr_file:N 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.

```

257 </2ekernel>
258 <*2ekernel | latexrelease>
259 <latexrelease>\IncludeInRelease{2020/10/01}%
260 <latexrelease> \@@_set_curr_file:N}{Set curr file}%
261 \ExplSyntaxOn
262 <@@=filehook>
263 \cs_new_protected:Npn \__filehook_set_curr_file:N #1
264 { \exp_args:NV \__filehook_set_curr_file:nNN #1 }
265 \cs_new_protected:Npn \__filehook_set_curr_file:nNN #1
266 {
267 \__filehook_file_parse_full_name:nN {#1}
268 \__filehook_set_curr_file_assign:nnnNN
269 }
270 \cs_new_protected:Npn \__filehook_set_curr_file_assign:nnnNN #1 #2 #3 #4 #5
271 {
272 \str_set:Nn #5 {#1}
273 \str_set:Nn #4 {#2#3}
274 }
275 \ExplSyntaxOff
276 </2ekernel | latexrelease>
277 <latexrelease>\EndIncludeInRelease
278 <*2ekernel>

```

(End definition for \@@_set_curr_file:N, \@@_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 Start by sanitising the file with __filehook_file_parse_full_name:nN then do __filehook_normalize_file_name:w filehook_file_subst_begin:nnn{<path>}{<name>}{<ext>}. _filehook_file_name_compose:nnn

```

279 </2ekernel>
280 <*2ekernel | latexrelease>
281 <latexrelease>\IncludeInRelease{2020/10/01}%
282 <latexrelease> {\_filehook_resolve_file_subst:w}{Replace files detect loops}%
283 \ExplSyntaxOn
284 \cs_new:Npn \__filehook_resolve_file_subst:w #1 \@nil
285 { \__filehook_file_parse_full_name:nN {#1} \__filehook_file_subst_begin:nnn }
286 \cs_new:Npn \__filehook_normalize_file_name:w #1 \@nil

```

```

287 { \_filehook_file_parse_full_name:nN {#1} \_filehook_file_name_compose:nnn }
288 \cs_new:Npn \_filehook_file_name_compose:nnn #1 #2 #3
289 { \tl_if_empty:nF {#1} { #1 / } #2#3 }

```

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 optimising).

```

290 \flag_new:n { \_filehook_file_replaced }
291 \cs_new:Npn \_filehook_if_file_replaced:TF #1 #2
292 { \flag_if_raised:nTF { \_filehook_file_replaced } {#1} {#2} }

```

First off, start by checking if the current file ($\langle name \rangle + \langle ext \rangle$) has a declared substitution. If not, then just put that as the name (including a possible $\langle path \rangle$ 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.

```

293 \cs_new:Npn \_filehook_file_subst_begin:nnn #1 #2 #3
294 {
295   \_filehook_file_subst_tortoise_hare:nn { #2#3 } { #2#3 }
296   { \_filehook_file_name_compose:nnn {#1} {#2} {#3} }
297 }
298 \ExplSyntaxOff
299 </2ekernel|latexrelease>
300 <latexrelease>\EndIncludeInRelease
301 <*2ekernel>

```

2.7.1 The Tortoise and Hare algorithm

If there is a substitution ($\langle true \rangle$ 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 $\langle file \rangle$ 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 $\langle false \rangle$ and the algorithm will end;

otherwise it will run until the Hare reaches the same spot as the tortoise and a loop is detected.

```

302 </2ekernel>
303 <*2ekernel|latexrelease>
304 <latexrelease>\IncludeInRelease{2020/10/01}%
305 <latexrelease>{__filehook_file_subst_tortoise_hare:nn}{Tortoise and Hare}%
306 \ExplSyntaxOn
307 \cs_new:Npn __filehook_file_subst_tortoise_hare:nn #1 #2 #3
308 {
309   \cs_if_exist:cTF { @file-subst@ #2 }
310   {
311     \flag_if_raised:nF { __filehook_file_replaced }
312     { \flag_raise:n { __filehook_file_replaced } }
313     \cs_if_exist:cTF { @file-subst@ \use:c { @file-subst@ #2 } }
314     {
315       __filehook_file_subst_loop:cc
316       { @file-subst@ #1 }
317       { @file-subst@ \use:c { @file-subst@ #2 } }
318     }
319     { \use:c { @file-subst@ #2 } }
320   }
321   { #3 }
322 }

```

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.

```

323 \cs_new:Npn __filehook_file_subst_loop:NN #1 #2
324 {
325   \token_if_eq_meaning:NNTF #1 #2
326   {
327     .tex
328     __filehook_file_subst_cycle_error:cN { @file-subst@ #1 } #1
329   }
330   { __filehook_file_subst_tortoise_hare:nn {#1} {#2} {#2} }
331 }
332 \cs_generate_variant:Nn __filehook_file_subst_loop:NN { cc }

```

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.

```

333 \cs_new:Npn __filehook_file_subst_cycle_error:NN #1 #2
334 {
335   __kernel_msg_expandable_error:nnff { kernel } { file-cycle }
336   {#1} { \use:c { @file-subst@ #1 } }
337   \token_if_eq_meaning:NNTF #1 #2
338   { __filehook_file_subst_cycle_error:cN { @file-subst@ #1 } #2 }
339 }
340 \cs_generate_variant:Nn __filehook_file_subst_cycle_error:NN { c }

```

And the error message:

```

341 __kernel_msg_new:nnn { kernel } { file-cycle }
342 { File~loop!~#1~replaced~by~#2... }

```


(End definition for `_filehook_resolve_file_subst:w` and others.)

```

343 \ExplSyntaxOff
344 \</2ekernel | latexrelease>
345 \<latexrelease>\EndIncludeInRelease
346 \<*2ekernel>
347 \<@@=

```

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.

```

348 \</2ekernel>
349 \<*2ekernel | latexrelease>
350 \<latexrelease>\IncludeInRelease{2020/10/01}%
351 \<latexrelease>{\disable@package@load}{Disable packages}%
352 \def\disable@package@load#1#2{%
353   \global\@namedef{\@pkg-disable@#1.\@pkgextension}{#2}}
354 \def\@disable@package@load@do#1#2{%
355   \@ifundefined{\@pkg-disable@#1}{#2}%
356   {\@nameuse{\@pkg-disable@#1}}}

```

`\reenable@package@load` undefines `\@pkg-disable@<package>` to realow loading a package.

```

357 \def\reenable@package@load#1{%
358   \global\expandafter\let
359   \csname \@pkg-disable@#1.\@pkgextension \endcsname \@undefined}
360 \</2ekernel | latexrelease>
361 \<latexrelease>\EndIncludeInRelease
362 \<latexrelease>\IncludeInRelease{0000/00/00}%
363 \<latexrelease>{\disable@package@load}{Disable packages}%
364 \<latexrelease>
365 \<latexrelease>\let\disable@package@load \@undefined
366 \<latexrelease>\let\@disable@package@load@do\@undefined
367 \<latexrelease>\let\reenable@package@load \@undefined
368 \<latexrelease>\EndIncludeInRelease
369 \<*2ekernel>

```

(End definition for `\disable@package@load`, `\reenable@package@load`, and `\@disable@package@load@do`. These functions are documented on page 5.)

2.9 High-level interfaces for L^AT_EX

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) L^AT_EX_{2 ϵ} names to allow for internal commands to be used outside this module (and in parts that still use L^AT_EX_{2 ϵ} syntax. We have to unset the @@ since we want double “at” sign in place of double underscores.

```

370 <@@=
371 </2ekernel>
372 <*2ekernel | latexrelease>
373 <latexrelease> \IncludeInRelease{2020/10/01}%
374 <latexrelease> { \@expl@@@filehook@if@no@extension@@nTF}{2e tmp interfaces}%
375 \ExplSyntaxOn

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

378 \cs_new_eq:NN \@expl@@@filehook@set@curr@file@@N
379             \__filehook_set_curr_file:N

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

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

384 \cs_new_eq:NN \@expl@@@filehook@if@file@replaced@@TF
385             \__filehook_if_file_replaced:TF
386

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

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

391 \cs_new_eq:NN \@expl@@@filehook@file@pop@@
392             \__filehook_file_pop:

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

395 \ExplSyntaxOff

396 </2ekernel | latexrelease>
397 <latexrelease> \EndIncludeInRelease
398 <*2ekernel>

```

This ends the kernel code in this file.

```

399 </2ekernel>

```

3 A sample package for structuring the log output

```

400 <*structuredlog>
401 <@@=filehook>

402 \ProvidesExplPackage
403     {structuredlog}{\ltfilehookdate}{\ltfilehookversion}
404     {Structuring the TeX transcript file}

\g_filehook_nesting_level_int Stores the current package nesting level.
405 \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.

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

(End 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.

```
408 \cs_new_protected:Npn \__filehook_log_file_record:n #1
409 {
410   \str_if_eq:nnT {#1} {START} { \int_gincr:N \g__filehook_nesting_level_int }
411   \iow_term:x
412   {
413     \prg_replicate:nn { \g__filehook_nesting_level_int } { = } ~
414     ( LEVEL ~ \int_use:N \g__filehook_nesting_level_int \c_space_tl #1 ) ~
415     \CurrentFileUsed
```

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

```
416     \str_if_eq:nnF \CurrentFileUsed \CurrentFile
417     { ~ ( \CurrentFile \c_space_tl requested ) }
418     \iow_newline:
419   }
420   \str_if_eq:nnT {#1} {STOP} { \int_gdecr:N \g__filehook_nesting_level_int }
421 }
```

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

```
422 \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.

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

(End definition for `__filehook_log_file_record:n`.)

```
425 <@@=>
426 </structuredlog>
```

4 Package emulations

4.1 Package `atveryend` emulation

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

```
427 <*2ekernel>
428 \declare@file@substitution{atveryend.sty}{atveryend-ltx.sty}
429 </2ekernel>
```

Here is the package file we point to:

```

430 <*atveryend-ltx>
431 \ProvidesPackage{atveryend-ltx}
432 [2020/08/19 v1.0a
433   Emulation of the original atvery package^^Jwith kernel methods]
   Here are new definitions for its interfaces now pointing to the hooks in \enddocument
434 \newcommand\AfterLastShipout {\AddToHook{enddocument/afterlastpage}}
435 \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.
436 \newcommand\AtEndAfterFileList{\AddToHook{enddocument/info}}
437 \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.
438 \ExplSyntaxOn
439 \newcommand\BeforeClearDocument[1]
440 { \AtEndDocument{#1}
441   \atveryend@DEPRECATED{BeforeClearDocument \tl_to_str:n{#1}}
442 }
443 \cs_new:Npn\atveryend@DEPRECATED #1
444 { \iow_term:x{=====~DEPRECATED~USAGE~#1~=====}}
445 \ExplSyntaxOff

(End definition for \BeforeClearDocument. This function is documented on page ??.)
446 </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.

Symbols		\AtVeryVeryEnd 437	
@@ commands:		B	
\@@_set_curr_file:N	<u>257</u>	\BeforeClearDocument	<u>438</u>
\@@_set_curr_file:nNN	<u>257</u>	\begingroup	199, 219
\@@_set_curr_file_assign:nnnNN	<u>257</u>	C	
\<filename>	<u>13</u>	class/after	3
A		class/after/.	3
\AddToHook	422, 424, 434, 435, 436, 437	class/before	3
\AddToHookNext	19, 423	class/before/.	3
\AfterLastShipout	434	\clearpage	3, 4
\AtEndAfterFileList	436	cs commands:	
\AtEndDocument	440	\cs:w	147, 156, 164, 165
\AtEndOfClass	3	\cs_end:	147, 156, 164, 165
\AtEndOfPackage	3	\cs_generate_variant:Nn	332, 340
\AtVeryEndDocument	435	\cs_gset:Npn	169

`\cs_gset:Npx` 149
`\cs_if_exist:NTF` 15, 309, 313
`\cs_new:Npn`
 ... 30, 38, 44, 57, 161, 167, 284,
 286, 288, 291, 293, 307, 323, 333, 443
`\cs_new_eq:NN` 179, 180, 376,
 378, 380, 382, 384, 387, 389, 391, 393
`\cs_new_protected:Npn` 49,
 61, 69, 78, 144, 153, 263, 265, 270, 408
`\cs_set:Npx` 147, 156
`\cs_undefine:N` 158
`\csname` 11, 14, 201, 202, 225, 240, 242, 359
`\csname\endcsname` 11, 12, 13
`\CurrentFile` 2,
 2, 2, 2, 3, 4, 7, 9, 9, 13, 13, 66,
 83, 105, 112, 115, 248, 252, 416, 417
`\CurrentFilePath` 2,
 2, 3, 7, 9, 13, 66, 82, 104, 249, 252
`\CurrentFilePathUsed`
 ... 2, 3, 65, 80, 102, 246, 249
`\CurrentFileUsed` 2, 3,
 4, 9, 13, 65, 81, 103, 246, 248, 415, 416

D

`\DeclareRobustCommand` 95, 125
`\def` 133, 198,
 201, 218, 238, 241, 244, 352, 354, 357

E

`\edef` 100
`\else` 243, 250
 else commands:
 `\else:` 168
`\endcsname` 11,
 12, 13, 201, 202, 225, 240, 242, 359
`\enddocument` 19, 19
`\endgroup` 13, 213, 226
`\EndIncludeInRelease` . 14, 23, 87, 120,
 130, 136, 172, 183, 190, 215, 228,
 232, 255, 277, 300, 345, 361, 368, 397
`\escapechar` 200, 220
 exp commands:
 `\exp_after:wN` 72, 163, 164
 `\exp_args:Ne` 46
 `\exp_args:Nf` 32, 34
 `\exp_args:Nv` 53, 264
 `\exp_not:n` 147, 156
`\expandafter` 11, 106, 128, 201,
 202, 222, 223, 224, 239, 240, 242, 358
`\ExplSyntaxOff` 12, 85,
 170, 181, 193, 275, 298, 343, 395, 445
`\ExplSyntaxOn` 7,
 29, 143, 178, 261, 283, 306, 375, 438

F

`\fi` 210, 211, 243, 253
 fi commands:
 `\fi:` 168
 file commands:
 `\file_full_name:n` 9, 35
 `\file_parse_full_name_apply:nN` ..
 ... 32, 47
 `\l_file_search_path_seq` 2, 9, 13
 file/after 1
 file/after/... 1
 file/before 1
 file/before/... 1
 filehook internal commands:
 `__filehook_drop_extension:N` 44, 388
 `__filehook_drop_extension_-`
 aux:nnn 54, 57
 `__filehook_file_name_compose:nnn`
 ... 279, 296
 `__filehook_file_parse_full_-`
 name:nN 7,
 7, 14, 28, 30, 53, 165, 267, 285, 287
 `__filehook_file_pop:` 59, 392
 `__filehook_file_pop_assign:nnnn`
 ... 59, 394
 `__filehook_file_push:` 59, 390
 `__filehook_file_subst_begin:nnn`
 ... 14, 285, 293
 `__filehook_file_subst_cycle_-`
 error:NN 16, 328, 333
 `__filehook_file_subst_loop:NN` ..
 ... 15, 302
 `__filehook_file_subst_tortoise_-`
 hare:nn 14, 15, 295, 302
 `__filehook_full_name:nn` 30
 `__filehook_if_file_replaced:TF` .
 ... 290, 385
 `__filehook_if_no_extension:nTF` .
 ... 44, 377
 `\g__filehook_input_file_seq` .. 7, 59
 `\l_filehook_internal_tl` 59
 `__filehook_log_file_record:n` .. 408
 `\g__filehook_nesting_level_int` ..
 ... 405, 410, 413, 414, 420
 `__filehook_normalize_file_-`
 name:w 279, 383
 `__filehook_resolve_file_subst:w`
 ... 279, 381
 `__filehook_set_curr_file:N` 263, 379
 `__filehook_set_curr_file:nNN` ...
 ... 264, 265
 `__filehook_set_curr_file_-`
 assign:nnnNN 30, 268, 270
 `__filehook_subst_add:nn` 142, 144, 179

_filehook_subst_empty_name_- chk:NN	144	_kernel_msg_new:nnn	341
_filehook_subst_file_normalize:n	144	L	
_filehook_subst_remove:n .	144, 180	\let	18, 19, 20, 21, 94, 187, 188, 212, 231, 248, 249, 358, 365, 366, 367
_filehook_substitution_- lthooadd:nn	10	\LoadPackageWithOptions	2
flag internal commands:		\long	133
\flag_filehook_file_replaced ..	15	\ltfilehookdate	403
flag_filehook_file_replaced ..	290	\ltfilehookversion	403
flag commands:		M	
\flag_if_raised:nTF	292, 311	msg commands:	
\flag_new:n	290	\msg_error:nnn	74
\flag_raise:n	312	N	
G		\newcommand	434, 435, 436, 437, 439
\global	212, 353, 358	O	
group commands:		\openin	1
\group_begin:	146, 155	P	
\group_end:	151, 159	package/after	3
I		package/after/.	3
if commands:		package/before	3
\if_meaning:w	168	package/before/.	3
\IfFileExists	96, 126, 134	prg commands:	
\ifx	11, 242, 247	\prg_replicate:nn	413
\include	1, 3, 4, 7	\ProvidesExplPackage	402
include/after	4	\ProvidesPackage	431
include/after/.	4	R	
include/before	4	\relax	12
include/before/.	4	\RequirePackage	2
include/end	4	S	
include/end/.	4	seq commands:	
\IncludeInRelease	5, 15, 27, 91, 122, 131, 141, 176, 184, 196, 216, 229, 236, 259, 281, 304, 350, 362, 373	\seq_gpop:NNTF	71
\includeonly	3, 4	\seq_gpush:Nn	63
\input	1, 2, 2, 6, 7, 12	\seq_new:N	60
\InputIfFileExists	7, 8, 8, 90	str commands:	
int commands:		\str_if_eq:NNTF	416
\int_gdecr:N	420	\str_if_eq:nnTF	410, 420
\int_gincr:N	410	\str_set:Nn	272, 273
\int_gset:Nn	406	\string	11, 224, 243
\int_new:N	405	T	
\int_set:Nn	148, 157	TeX and L ^A T _E X 2 _ε commands:	
\int_use:N	414	\@input	113, 129, 135
iow commands:		\@addtofilelist	110, 129, 135
\iow_newline:	418	\@curr@file ..	11, 12, 13, 194, 245, 247
\iow_term:n	411, 444	\@curr@file@reqd .	12, 13, 194, 247, 251
K		\@currnamestack	18, 407
kernel internal commands:		\@disable@packageload@do	348
_kernel_msg_expandable_- error:nnnn	335	\@empty	225
		\@expl@@@filehook@drop@extension@@N	12, 210, 211, 387
		\@expl@@@filehook@file@pop@@	117, 391

