

The main structure of documents

Frank Mittelbach Chris Rowley Alan Jeffrey
David Carlisle

2018/11/11

This file is maintained by the L^AT_EX Project team.
Bug reports can be opened (category `latex`) at
<https://latex-project.org/bugs.html>.

1 Introduction

This file implements the following declarations, which replace `\documentstyle` in L^AT_EX 2_ε documents.

Note that old documents containing `\documentstyle` will be run using a compatibility option—thus keeping everyone happy, we hope!

The overall idea is that there are two types of ‘style files’: ‘class files’ which define elements and provide a default formatting for them; and ‘packages’ which provide extra functionality. One difference between L^AT_EX 2_ε and L^AT_EX 2.09 is that L^AT_EX 2_ε packages may have options. Note that options to classes/packages may be implemented such that they input files, but these file names are not necessarily directly related to the option name.

2 User interface

```
\documentclass[main-option-list]{class}[version]
```

There must be exactly one such declaration, and it must come first. The *main-option-list* is a list of options which can modify the formatting of elements which are defined in the *class* file as well as in all following `\usepackage` declarations (see below). The *version* is a version number, beginning with a date in the format YYYY/MM/DD. If an older version of the class is found, a warning is issued.

```
\documentstyle[main-option-list]{class}[version]
```

The `\documentstyle` declaration is kept in order to maintain upward compatibility with L^AT_EX 2.09 documents. It is similar to `\documentclass`, but it causes all options in *main-option-list* that the *class* does not use to be passed to

`\RequirePackage` after the options have been processed. This maintains compatibility with the 2.09 behaviour. Also a flag is set to indicate that the document is to be processed in L^AT_EX 2.09 compatibility mode. As far as most packages are concerned, this only affects the warnings and errors L^AT_EX generates. This flag does affect the definition of font commands, and `\sloppy`.

`\usepackage` [*package-option-list*] {*package-list*} [*version*]

There can be any number of these declarations. All packages in *package-list* are called with the same options.

Each *package* file defines new elements (or modifies those defined in the *class*), and thus extends the range of documents which can be processed. The *package-option-list* is a list of options which can modify the formatting of elements defined in the *package* file. The *version* is a version number, beginning with a date in the format YYYY/MM/DD. If an older version of the package is found, a warning is issued.

Each package is loaded only once. If the same package is requested more than once, nothing happens, unless the package has been requested with options that were not given the first time it was loaded, in which case an error is produced.

As well as processing the options given in the *package-option-list*, each package processes the *main-option-list*. This means that options that affect all of the packages can be given globally, rather than repeated for every package.

Note that class files have the extension `.cls`, packages have the extension `.sty`.

`filecontents`

The environment `filecontents` is intended for passing the contents of packages, options, or other files along with a document in a single file. It has one argument, which is the name of the file to create. If that file already exists (maybe only in the current directory if the OS supports a notion of a ‘current directory’ or ‘default directory’) then nothing happens (except for an information message) and the body of the environment is bypassed. Otherwise, the body of the environment is written verbatim to the file name given as the first argument, together with some comments about how it was produced.

The environment is allowed only before `\documentclass` to ensure that all packages or options necessary for this particular run are present when needed. The begin and end tags should each be on a line by itself. There is also a star-form; this does not write extra comments into the file.

2.1 Option processing

When the options are processed, they are divided into two types: *local* and *global*:

- For a class, the options in the `\documentclass` command are local.
- For a package, the options in the `\usepackage` command are local, and the options in the `\documentclass` command are global.

The options for `\documentclass` and `\usepackage` are processed in the following way:

1. The local and global options that have been declared (using `\DeclareOption` as described below) are processed first.

In the case of `\ProcessOptions`, they are processed in the order that they were declared in the class or package.

In the case of `\ProcessOptions*`, they are processed in the order that they appear in the option-lists. First the global options, and then the local ones.

2. Any remaining local options are dealt with using the default option (declared using the `\DeclareOption*` declaration described below). For document classes, this usually does nothing, but records the option on a list of unused options. For packages, this usually produces an error.

Finally, when `\begin{document}` is reached, if there are any global options which have not been used by either the class or any package, the system will produce a warning.

3 Class and Package interface

3.1 Class name and version

`\ProvidesClass` A class can identify itself with the `\ProvidesClass{<name>}[<version>]` command. The *<version>* should begin with a date in the format YYYY/MM/DD.

3.2 Package name and version

`\ProvidesPackage` A package can identify itself with the `\ProvidesPackage{<name>}[<version>]` command. The *<version>* should begin with a date in the format YYYY/MM/DD.

3.3 Requiring other packages

`\RequirePackage` Packages or classes can load other packages using `\RequirePackage[<options>]{<name>}[<version>]`. If the package has already been loaded, then nothing happens unless the requested options are not a subset of the options with which it was loaded, in which case an error is called.

`\LoadClass` Similar to `\RequirePackage`, but for classes, may not be used in package files.

`\PassOptionsToPackage` Packages can pass options to other packages using:

`\PassOptionsToPackage{<options>}{<package>}`.

`\PassOptionsToClass` This adds the *<options>* to the options list of any future `\RequirePackage` or `\usepackage` command. For example:

```
\PassOptionsToPackage{foo,bar}{fred}
\RequirePackage[baz]{fred}
```

is the same as:

```
\RequirePackage[foo,bar,baz]{fred}
```

`\LoadClassWithOptions` `\LoadClassWithOptions{<name>}[<version>]:`
This is similar to `\LoadClass`, but it always calls class `<name>` with exactly the same option list that is being used by the current class, rather than an option explicitly supplied or passed on by `\PassOptionsToClass`.
`\RequirePackageWithOptions` `\RequirePackageWithOptions` is the analogous command for packages.
This is mainly intended to allow one class to simply build on another, for example:

```
\LoadClassWithOptions{article}
```

This should be contrasted with the slightly different construction

```
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
\ProcessOptions
\LoadClass{article}
```

As used here, the effects are more or less the same, but the version using `\LoadClassWithOptions` is slightly quicker (and less to type). If, however, the class declares options of its own then the two constructions are different; compare, for example:

```
\DeclareOption{landscape}{...}
\ProcessOptions
\LoadClassWithOptions{article}
```

with:

```
\DeclareOption{landscape}{...}
\DeclareOption*{\PassOptionsToClass{\CurrentOption}{article}}
\ProcessOptions
\LoadClass{article}
```

In the first case, the article class will be called with option `landscape` precisely when the current class is called with this option; but in the second example it will not as in that case article is only passed options by the default option handler, which is not used for `landscape` as that option is explicitly declared.

`\@ifpackageloaded` To find out if a package has already been loaded, use
`\@ifclassloaded` `\@ifpackageloaded{<package>}{<true>}{<false>}`.
`\@ifpackagelater` To find out if a package has already been loaded with a version equal to or more recent than `<version>`, use
`\@ifclasslater` `\@ifpackagelater{<package>}{<version>}{<true>}{<false>}`.
`\@ifpackagewith` To find out if a package has already been loaded with at least the options
`\@ifclasswith` `<options>`, use `\@ifpackagewith{<package>}{<options>}{<true>}{<false>}`.

There exists one package that can't be tested with the above commands: the `fontenc` package pretends that it was never loaded to allow for repeated reloading with different options (see `ltoutenc.dtx` for details).

3.4 Declaring new options

Options for classes and packages are built using the same macros.

`\DeclareOption` To define a builtin option, use `\DeclareOption{<name>}{<code>}`.
`\DeclareOption*` To define the default action to perform for local options which have not been declared, use `\DeclareOption*{<code>}`.

Note: there should be no use of

`\RequirePackage`, `\DeclareOption`, `\DeclareOption*` or `\ProcessOptions` inside `\DeclareOption` or `\DeclareOption*`.

Possible uses for `\DeclareOption*` include:

`\DeclareOption*{}`

Do nothing. Silently accept unknown options. (This suppresses the usual warnings.)

`\DeclareOption*{\@unkownoptionerror}`

Complain about unknown local options. (The initial setting for package files.)

`\DeclareOption*{\PassOptionsToPackage{\CurrentOption}{<pkg-name>}`

Handle the the current option by passing it on to the package `<pkg-name>`, which will presumably be loaded via `\RequirePackage` later in the file. This is useful for building ‘extension’ packages, that perhaps handle a couple of new options, but then pass everything else on to an existing package.

`\DeclareOption*{\InputIfFileExists{xx-\CurrentOption.yyy}%
 {}}%`
`{\OptionNotUsed}`

Handle the option `foo` by loading the file `xx-foo.yyy` if it exists, otherwise do nothing, but declare that the option was not used. Actually the `\OptionNotUsed` declaration is only needed if this is being used in class files, but does no harm in package files.

3.5 Safe Input Macros

`\InputIfFileExists` `\InputIfFileExists{<file>}{<then>}{<else>}`
 Inputs `<file>` if it exists. Immediately before the input, `<then>` is executed. Otherwise `<else>` is executed.

`\IfFileExists` As above, but does not input the file.

One thing you might like to put in the `<else>` clause is

`\@missingfileerror` This starts an interactive request for a filename, supplying default extensions. Just hitting return causes the whole input to be skipped and entering `x` quits the current run,

`\input` This has been redefined from the L^AT_EX2.09 definition, in terms of the new commands `\InputIfFileExists` and `\@missingfileerror`.

`\listfiles` Giving this declaration in the preamble causes a list of all files input via the ‘safe input’ commands to be listed at the end. Any strings specified in the optional argument to `\ProvidesPackage` are listed alongside the file name. So files in standard (and other non-standard) distributions can put informative strings in this argument.

4 Implementation

	1 <code>*2ekernel</code>
<code>\if@compatibility</code>	The flag for compatibility mode. 2 <code>\newif\if@compatibility</code>
<code>\@documentclasshook</code>	The hook called after the first <code>\documentclass</code> command. By default this checks to see if <code>\@normalsize</code> is undefined, and if so, sets it to <code>\normalsize</code> . 3 <code>\def\@documentclasshook{%</code> 4 <code> \ifx\@normalsize\undefined</code> 5 <code> \let\@normalsize\normalsize</code> 6 <code> \fi</code> 7 <code>}</code>
<code>\@declaredoptions</code>	This list is automatically built by <code>\DeclareOption</code> . It is the list of options (separated by commas) declared in the class or package file and it defines the order in which the the corresponding <code>\ds@<option></code> commands are executed. All local <code><option></code> s which are not declared will be processed in the order defined by the optional argument of <code>\documentclass</code> or <code>\usepackage</code> . 8 <code>\let\@declaredoptions\@empty</code>
<code>\@classoptionslist</code>	List of options of the main class. 9 <code>\let\@classoptionslist\relax</code> 10 <code>\@onlypreamble\@classoptionslist</code>
<code>\@unusedoptionlist</code>	List of options of the main class that haven't been declared or loaded as class option files. 11 <code>\let\@unusedoptionlist\@empty</code> 12 <code>\@onlypreamble\@unusedoptionlist</code>
<code>\CurrentOption</code>	Name of current package or option. 13 <code>\let\CurrentOption\@empty</code>
<code>\@currname</code>	Name of current package or option. 14 <code>\let\@currname\@empty</code>
<code>\@currentext</code>	The current file extension. 15 <code>\global\let\@currentext=\@empty</code>
<code>\@clsextension</code>	The two possible values of <code>\@currentext</code> .
<code>\@pkgextension</code>	16 <code>\def\@clsextension{cls}</code> 17 <code>\def\@pkgextension{sty}</code> 18 <code>\@onlypreamble\@clsextension</code> 19 <code>\@onlypreamble\@pkgextension</code>

```

\@pushfilename Commands to push and pop the file name and extension.
\@popfilename #1 current name.
\@currnamestack #2 current extension.
                #3 current catcode of @.
                #4 Rest of the stack.
20 \def\@pushfilename{%
21   \xdef\@currnamestack{%
22     {\@currname}%
23     {\@current}%
24     {\the\catcode'\@}%
25     \@currnamestack}}
26 \@onlypreamble\@pushfilename

27 \def\@popfilename{\expandafter\@p@pfilename\@currnamestack\@nil}
28 \@onlypreamble\@popfilename

29 \def\@p@pfilename#1#2#3#4\@nil{%
30   \gdef\@currname{#1}%
31   \gdef\@current{#2}%
32   \catcode'\@#3\relax
33   \gdef\@currnamestack{#4}}
34 \@onlypreamble\@p@pfilename

35 \gdef\@currnamestack{}
36 \@onlypreamble\@currnamestack

\@optionlist Returns the option list of the file.
37 \def\@optionlist#1{%
38   \@ifundefined{opt@#1}\@empty{\csname opt@#1\endcsname}}
39 \@onlypreamble\@optionlist

\@ifpackageloaded \@ifpackageloaded{<name>} Checks to see whether a file has been loaded.
\@ifclassloaded  40 \def\@ifpackageloaded{\@ifl@aded\@pkgextension}
                  41 \def\@ifclassloaded{\@ifl@aded\@clsextension}
                  42 \@onlypreamble\@ifpackageloaded
                  43 \@onlypreamble\@ifclassloaded

                  44 \def\@ifl@aded#1#2{%
                  45   \expandafter\ifx\csname ver@#2.#1\endcsname\relax
                  46     \expandafter\@secondoftwo
                  47   \else
                  48     \expandafter\@firstoftwo
                  49   \fi}
                  50 \@onlypreamble\@ifl@aded

\@ifpackagelater \@ifpackagelater{<name>}{YYYY/MM/DD} Checks that the package loaded is
\@ifclasslater   \@ifclasslater more recent than the given date.
                  51 \def\@ifpackagelater{\@ifl@ater\@pkgextension}
                  52 \def\@ifclasslater{\@ifl@ater\@clsextension}
                  53 \@onlypreamble\@ifpackagelater
                  54 \@onlypreamble\@ifclasslater

```

```

55 \def\@ifl@ter#1#2{%
56   \expandafter\@ifl@t@r
57     \csname ver@#2.#1\endcsname}
58 \@onlypreamble\@ifl@ter
59 </2ekernel>

    This internal macro is also used in \NeedsTeXFormat.

60 <latexrelease>\IncludeInRelease{2018/04/01}%
61 <latexrelease>           {\@ifl@t@r}{Guard against bad input}%
62 <*2ekernel | latexrelease>
63 \def\@ifl@t@r#1#2{%
64   \ifnum\expandafter\@parse@version@#1//00\@nil<%
65     \expandafter\@parse@version@#2//00\@nil
66     \expandafter\@secondoftwo
67   \else
68     \expandafter\@firstoftwo
69   \fi}
70 \def\@parse@version@#1{\@parse@version@#1}
71 </2ekernel | latexrelease>
72 <latexrelease>\EndIncludeInRelease
73 <latexrelease>\IncludeInRelease{0000/00/00}%
74 <latexrelease>           {\@ifl@t@r}{Guard against bad input}%
75 <latexrelease>\def\@ifl@t@r#1#2{%
76 <latexrelease>  \ifnum\expandafter\@parse@version@#1//00\@nil<%
77 <latexrelease>    \expandafter\@parse@version@#2//00\@nil
78 <latexrelease>    \expandafter\@secondoftwo
79 <latexrelease>    \else
80 <latexrelease>    \expandafter\@firstoftwo
81 <latexrelease>    \fi}
82 <latexrelease>\let\@parse@version@\@undefined
83 <latexrelease>\EndIncludeInRelease
84 <*2ekernel>

85 \@onlypreamble\@ifl@t@r
86 </2ekernel>
87 <*2ekernel | latexreleasefirst>
88 \def\@parse@version@#1/#2/#3#4#5\@nil{%
89 \@parse@version@dashed#1-#2-#3#4\@nil
90 }

```

The \if test here ensures that an argument with no / or - produces 0 (actually 00).

```

91 \def\@parse@version@dashed#1-#2-#3#4#5\@nil{%
92   \if\relax#2\relax\else#1\fi#2#3#4 }
93 </2ekernel | latexreleasefirst>
94 <*2ekernel>

```

`\@ifpackagewith` `\@ifclasswith` `\@ifpackagewith{<name>}{<option-list>}` Checks that <option-list> is a subset of the options **with** which <name> was loaded.

```

95 \def\@ifpackagewith{\@if@options\@pkgextension}

```



```

96 \def\@ifclasswith{\@ifoptions\clsextension}
97 \@onlypreamble\ifpackagewith
98 \@onlypreamble\ifclasswith
99 \def\@ifoptions#1#2{%
100 \expandafter\@ifpti@ns{\optionlist{#2.#1}}
101 \@onlypreamble\ifoptions
    Probably shouldn't use \CurrentOption here... (changed to \reserved@b.)
102 </2ekernel>
103 <latexrelease>\IncludeInRelease{2017/01/01}%
104 <latexrelease>          {\@ifpti@ns}{Spaces in option clash check}%
105 <*2ekernel | latexrelease>
106 \def\@ifpti@ns#1#2{%
107 \let\reserved@a\@firstoftwo
108 \edef\reserved@b{\zap@space#2 \@empty}%
109 \@for\reserved@b:=\reserved@b\do{%
110 \ifx\reserved@b\@empty
111 \else
112 \expandafter\in@\expandafter{\expandafter,\reserved@b,}{, #1,}%
113 \ifin@
114 \else
115 \let\reserved@a\@secondoftwo
116 \fi
117 \fi
118 }%
119 \reserved@a}
120 </2ekernel | latexrelease>
121 <latexrelease>\EndIncludeInRelease
122 <latexrelease>\IncludeInRelease{0000/00/00}%
123 <latexrelease>          {\@ifpti@ns}{Spaces in option clash check}%
124 <latexrelease>\def\@ifpti@ns#1#2{%
125 <latexrelease> \let\reserved@a\@firstoftwo
126 <latexrelease> \@for\reserved@b:=#2\do{%
127 <latexrelease> \ifx\reserved@b\@empty
128 <latexrelease> \else
129 <latexrelease> \expandafter\in@\expandafter
130 <latexrelease>          {\expandafter,\reserved@b,}{, #1,}%
131 <latexrelease> \ifin@
132 <latexrelease> \else
133 <latexrelease> \let\reserved@a\@secondoftwo
134 <latexrelease> \fi
135 <latexrelease> \fi
136 <latexrelease> }%
137 <latexrelease> \reserved@a}
138 <latexrelease>\EndIncludeInRelease
139 <*2ekernel>
140 \@onlypreamble\@ifpti@ns

```

`\ProvidesPackage` Checks that the current filename is correct, and defines `\ver@filename`.

```

141 \def\ProvidesPackage#1{%
142   \xdef\@gtempa{#1}%
143   \ifx\@gtempa\@currname\else
144     \@latex@warning@no@line{You have requested
145       \@cls@pkg\space'\@currname',\MessageBreak
146       but the \@cls@pkg\space provides '#1'}%
147   \fi
148   \@ifnextchar[\@pr@videpackage{\@pr@videpackage[]}]%
149   \@onlypreamble\ProvidesPackage

150 \def\@pr@videpackage[#1]{%
151   \expandafter\xdef\csname ver@\@currname.\@current\endcsname{#1}%
152   \ifx\@current\@clsextension
153     \typeout{Document Class: \@gtempa\space#1}%
154   \else
155     \wlog{Package: \@gtempa\space#1}%
156   \fi}
157 \@onlypreamble\@pr@videpackage

```

`\ProvidesClass` Like `\ProvidesPackage`, but for classes.

```

158 \let\ProvidesClass\ProvidesPackage
159 \@onlypreamble\ProvidesClass

```

`\ProvidesFile` Like `\ProvidesPackage`, but for arbitrary files. Do not apply `\@onlypreamble` to these, as we may want to label files input during the document.

`\@providesfile`

```

160 \def\ProvidesFile#1{%
161   \begingroup
162   \catcode'\ 10 %
163   \ifnum \endlinechar<256 %
164     \ifnum \endlinechar>\m@ne
165       \catcode\endlinechar 10 %
166   \fi
167   \fi
168   \@makeother\/%
169   \@makeother\&%
170   \kernel@ifnextchar[{\@providesfile{#1}}{\@providesfile{#1}[]}]

```

During `initex` a special version of `\@providesfile` is used. The real definition is installed right at the end, in `ltxfinal.dtx`.

```

\def\@providesfile#1[#2]{%
  \wlog{File: #1 #2}%
  \expandafter\xdef\csname ver@#1\endcsname{#2}%
  \endgroup}

```

`\PassOptionsToPackage` If the package has been loaded, we check that it was first loaded with the options.

`\PassOptionsToClass` Otherwise we add the option list to that of the package.

```

171 </2ekernel)

```

```

172 <latexrelease>\IncludeInRelease{2018/12/01}%
173 <latexrelease>          {\@pass@ptions}{Raw option lists}%
174 <*2ekernel | latexrelease>
175 \def\@pass@ptions#1#2#3{%
176   \expandafter\xdef\csname opt@#3.#1\endcsname{%
177     \ifundefined{opt@#3.#1}\@empty
178     {\csname opt@#3.#1\endcsname,}%
179     \zap@space#2 \@empty}%
    Extend raw option list
180   \ifundefined{@raw@opt@#3.#1}%
181     {\expandafter\gdef\csname @raw@opt@#3.#1\endcsname{#2}}%
182     {\expandafter\g@addto@macro\csname @raw@opt@#3.#1\endcsname{,#2}}%
183 }
184 </2ekernel | latexrelease>
185 <latexrelease>\EndIncludeInRelease
186 <latexrelease>\IncludeInRelease{0000/00/00}%
187 <latexrelease>          {\@pass@ptions}{Raw option lists}%
188 <latexrelease>\def\@pass@ptions#1#2#3{%
189 <latexrelease>   \expandafter\xdef\csname opt@#3.#1\endcsname{%
190 <latexrelease>     \ifundefined{opt@#3.#1}\@empty
191 <latexrelease>     {\csname opt@#3.#1\endcsname,}%
192 <latexrelease>     \zap@space#2 \@empty}}
193 <*2ekernel>
194 \@onlypreamble\@pass@ptions
195 \def\PassOptionsToPackage{\@pass@ptions\@pkgextension}
196 \def\PassOptionsToClass{\@pass@ptions\@clsextension}
197 \@onlypreamble\PassOptionsToClass

```

`\DeclareOption` Adds an option as a `\ds@` command, or the default `\default@ds` command.

```

\DeclareOption* 198 \def\DeclareOption{%
199   \let\@fileswith@ptions\@badrequireerror
200   \@ifstar\@defdefault@ds\@declareoption}
201 \long\def\@declareoption#1#2{%
202   \xdef\@declaredoptions{\@declaredoptions,#1}%
203   \toks@{#2}%
204   \expandafter\edef\csname ds@#1\endcsname{\the\toks@}}
205 \long\def\@defdefault@ds#1{%
206   \toks@{#1}%
207   \edef\default@ds{\the\toks@}}
208 \@onlypreamble\DeclareOption
209 \@onlypreamble\@declareoption
210 \@onlypreamble\@defdefault@ds

```

`\OptionNotUsed` If we are in a class file, add `\CurrentOption` to the list of unused options. Otherwise, in a package file do nothing.

```

211 </2ekernel>
212 <latexrelease>\IncludeInRelease{2018/12/01}%
213 <latexrelease>          {\OptionNotUsed}{filter unused option list}%

```

```

214 <*2ekernel | latexrelease>
215 \def\@remove@eq@value#1=#2\@nil{#1}

216 \def\OptionNotUsed{%
217   \ifx\@current\@clsextension
218     \xdef\@unusedoptionlist{%
219       \ifx\@unusedoptionlist\@empty\else\@unusedoptionlist,\fi
220       \expandafter\@remove@eq@value\CurrentOption=\@nil}%
221   \fi}
222 </2ekernel | latexrelease>
223 <latexrelease>\EndIncludeInRelease
224 <latexrelease>\IncludeInRelease{0000/00/00}%
225 <latexrelease>           {\OptionNotUsed}{filter unused option list}%
226 <latexrelease>\let\@remove@eq@value\undefined

227 <latexrelease>\def\OptionNotUsed{%
228 <latexrelease>   \ifx\@current\@clsextension
229 <latexrelease>     \xdef\@unusedoptionlist{%
230 <latexrelease>       \ifx\@unusedoptionlist\@empty\else\@unusedoptionlist,\fi
231 <latexrelease>       \CurrentOption}%
232 <latexrelease>   \fi}
233 <*2ekernel>

234 \@onlypreamble\OptionNotUsed

```

`\default@ds` The default default option code. Set by `\@onefilewithoptions` to either `\OptionNotUsed` for classes, or `\@unknownoptionerror` for packages. This may be reset in either case with `\DeclareOption*`.

```
235 % \let\default@ds\OptionNotUsed
```

`\ProcessOptions` `\ProcessOptions` calls `\ds@option` for each known package option, then calls `\default@ds` for each option on the local options list. Finally resets all the declared options to `\relax`. The empty option does nothing, this has to be reset on the off chance it's set to `\relax` if an empty element gets into the `\@declaredoptions` list.

The star form is similar but executes options given in the order specified in the document, not the order they are declared in the file. In the case of packages, global options are executed before local ones.

```

236 \def\ProcessOptions{%
237   \let\ds@\@empty
238   \edef\@curroptions{\@optionlist{\@currname.\@current}}%
239   \@ifstar\@xprocess@options\@process@options}
240 \@onlypreamble\ProcessOptions

241 \def\@process@options{%
242   \@for\CurrentOption:=\@declaredoptions\do{%
243     \ifx\CurrentOption\@empty\else
244       \@expandtwoargs\in@{,\CurrentOption,}%
245       ,\ifx\@current\@clsextension\else\@classoptionslist,\fi
246       \@curroptions,}%
247   \ifin@

```

```

248     \@use@option
249     \expandafter\let\csname ds@\CurrentOption\endcsname\@empty
250     \fi
251 \fi}%
252 \@process@pti@ns}
253 \@onlypreamble\@process@ptions
254 \def\@xprocess@ptions{%
255 \ifx\@current\@clsextension\else
256 \for\CurrentOption:=\@classoptionslist\do{%
257 \ifx\CurrentOption\@empty\else
258 \expandtwoargs\in@{\CurrentOption,}{\@declaredoptions,}%
259 \ifin@
260 \use@option
261 \expandafter\let\csname ds@\CurrentOption\endcsname\@empty
262 \fi
263 \fi}%
264 \fi
265 \@process@pti@ns}
266 \@onlypreamble\@xprocess@ptions

```

The common part of `\ProcessOptions` and `\ProcessOptions*`.

```

267 \def\@process@pti@ns{%
268 \for\CurrentOption:=\@curroptions\do{%
269 \ifundefined{ds@\CurrentOption}%
270 {\use@option
271 \default@ds}%

```

There should not be any non-empty definition of `\CurrentOption` at this point, as all the declared options were executed earlier. This is for compatibility with 2.09 styles which use `\def\ds@...` directly, and so have options which do not appear in `\@declaredoptions`.

```

272 \use@option}%

```

Clear all the definitions for option code. First set all the declared options to `\relax`, then reset the ‘default’ and ‘empty’ options. and the list of declared options.

```

273 \@for\CurrentOption:=\@declaredoptions\do{%
274 \expandafter\let\csname ds@\CurrentOption\endcsname\relax}%
275 \let\CurrentOption\@empty
276 \let\@fileswith@pti@ns\@fileswith@pti@ns
277 \AtEndOfPackage{\let\@unprocessedoptions\relax}}
278 \@onlypreamble\@process@pti@ns

```

`\@options` `\@options` is a synonym for `\ProcessOptions*` for upward compatibility with L^AT_EX 2.09 style files.

```

279 \def\@options{\ProcessOptions*}
280 \@onlypreamble\@options

```

`\@use@option` Execute the code for the current option.

```

281 </2ekernel>
282 <latexrelease>\IncludeInRelease{2018/12/01}%
283 <latexrelease>          {\use@option}{filter unused option list}%
284 <*2ekernel | latexrelease>
285 \def\use@option{%
286   \@expandtwoargs\@removeelement
287     {\expandafter\@remove@eq@value\CurrentOption=\@nil}%
288   \@unusedoptionlist\@unusedoptionlist
289   \csname ds@\CurrentOption\endcsname}
290 </2ekernel | latexrelease>
291 <latexrelease>\EndIncludeInRelease
292 <latexrelease>\IncludeInRelease{0000/00/00}%
293 <latexrelease>          {\use@option}{filter unused option list}%
294 <latexrelease>\def\use@option{%
295 <latexrelease>   \@expandtwoargs\@removeelement\CurrentOption
296 <latexrelease>   \@unusedoptionlist\@unusedoptionlist
297 <latexrelease>   \csname ds@\CurrentOption\endcsname}
298 <*2ekernel>
299 \@onlypreamble\use@option

```

`\ExecuteOptions` `\ExecuteOptions{<option-list>}` executes the code declared for each option.

```

300 </2ekernel>
301 <latexrelease>\IncludeInRelease{2017/01/01}%
302 <latexrelease>          {\ExecuteOptions}{Spaces in \ExecuteOptions}%
303 <*2ekernel | latexrelease>
304 \def\ExecuteOptions#1{%
    Use \@fortmp here as it is anyway cleared during \@for loop so does not change
    any existing names.
305   \edef\@fortmp{\zap@space#1 \@empty}%
306   \def\reserved@a##1\@nil{%
307     \@for\CurrentOption:=\@fortmp\do
308       {\csname ds@\CurrentOption\endcsname}%
309     \edef\CurrentOption{##1}}%
310   \expandafter\reserved@a\CurrentOption\@nil}
311 </2ekernel | latexrelease>
312 <latexrelease>\EndIncludeInRelease
313 <latexrelease>\IncludeInRelease{0000/00/00}%
314 <latexrelease>          {\ExecuteOptions}{Spaces in \ExecuteOptions}%
315 <latexrelease>\def\ExecuteOptions#1{%
316 <latexrelease>   \def\reserved@a##1\@nil{%
317 <latexrelease>     \@for\CurrentOption:=#1\do
318 <latexrelease>       {\csname ds@\CurrentOption\endcsname}%
319 <latexrelease>     \edef\CurrentOption{##1}}%
320 <latexrelease>   \expandafter\reserved@a\CurrentOption\@nil}
321 <latexrelease>\EndIncludeInRelease
322 <*2ekernel>
323 \@onlypreamble\ExecuteOptions

```

The top-level commands, which just set some parameters then call the internal command, `\@fileswithoptions`.

```
\documentclass The main new-style class declaration.
324 \def\documentclass{%
325   \let\documentclass\@twoclasseserror
326   \if@compatibility\else\let\usepackage\RequirePackage\fi
327   \@fileswithoptions\@clsextension}
328 \@onlypreamble\documentclass
```

```
\documentstyle 2.09 style class 'style' declaration.
329 \def\documentstyle{%
330   \makeatletter\input{latex209.def}\makeatother
331   \documentclass}
332 \@onlypreamble\documentstyle
```

```
\RequirePackage Load package if not already loaded.
333 \def\RequirePackage{%
334   \@fileswithoptions\@pkgextension}
335 \@onlypreamble\RequirePackage
```

```
\LoadClass Load class.
336 \def\LoadClass{%
337   \ifx\@currentx\@pkgextension
338     \@latex@error
339     {\noexpand\LoadClass in package file}%
340     {You may only use \noexpand\LoadClass in a class file.}%
341   \fi
342   \@fileswithoptions\@clsextension}
343 \@onlypreamble\LoadClass
```

```
\@loadwithoptions Pass the current option list on to a class or package. #1 is \@cls-or-pkgextension,
#2 is \RequirePackage or \LoadClass, #3 is the class or package to be loaded.
344 \def\@loadwithoptions#1#2#3{%
345   \expandafter\let\csname opt@#3.#1\expandafter\endcsname
346     \csname opt@\@currname.\@currentx\endcsname
347   #2{#3}}
348 \@onlypreamble\@loadwithoptions
```

```
\LoadClassWithOptions Load class '#1' with the current option list.
349 \def\LoadClassWithOptions{%
350   \@loadwithoptions\@clsextension\LoadClass}
351 \@onlypreamble\LoadClassWithOptions
```

```
\RequirePackageWithOptions Load package '#1' with the current option list.
352 \def\RequirePackageWithOptions{%
353   \AtEndOfPackage{\let\@unprocessedoptions\relax}%
354   \@loadwithoptions\@pkgextension\RequirePackage}
355 \@onlypreamble\RequirePackageWithOptions
```

`\usepackage` To begin with, `\usepackage` produces an error. This is reset by `\documentclass`.

```
356 \def\usepackage#1{%
357   \@latex@error
358     {\noexpand \usepackage before \string\documentclass}%
359     {\noexpand \usepackage may only appear in the document
360      preamble, i.e.,\MessageBreak
361      between \noexpand\documentclass and
362      \string\begin{document}.}%
363   \@gobble}
364 \@onlypreamble\usepackage
```

`\NeedsTeXFormat` Check that the document is running on the correct system.

```
365 \def\NeedsTeXFormat#1{%
366   \def\reserved@a{#1}%
367   \ifx\reserved@a\fmtname
368     \expandafter\@needsformat
369   \else
370     \@latex@error{This file needs format '\reserved@a'%
371       \MessageBreak but this is '\fmtname'}{%
372       The current input file will not be processed
373       further,\MessageBreak
374       because it was written for some other flavor of
375       TeX.\MessageBreak\@ehd}%
```

If the file is not meant to be processed by L^AT_EX 2_ε we stop inputting it, but we do not end the run. We just end inputting the current file.

```
376   \endinput \fi}
377 \@onlypreamble\NeedsTeXFormat

378 \def\@needsformat{%
379   \@ifnextchar[%
380     \@needsformat
381   {}]}
382 \@onlypreamble\@needsformat

383 \def\@needsformat[#1]{%
384   \@ifl@t@r\fmtversion{#1}{}%
385   {\@latex@warning@no@line
386     {You have requested release '#1' of LaTeX,\MessageBreak
387     but only release '\fmtversion' is available}}
388 \@onlypreamble\@needsformat
```

`\zap@space` `\zap@space foo<space>\@empty` removes all spaces from `foo` that are not protected by `{ }` groups.

```
389 \def\zap@space#1 #2{%
390   #1%
391   \ifx#2\@empty\else\expandafter\zap@space\fi
392   #2}
```

`\@fileswithoptions` The common part of `\documentclass` and `\usepackage`.


```

393 \def\@fileswithoptions#1{%
394   \@ifnextchar[%
395     {\@fileswithoptions#1}%
396     {\@fileswithoptions#1[]}}
397 \@onlypreamble\@fileswithoptions
398 \def\@fileswithoptions#1[#2]#3{%
399   \@ifnextchar[%
400     {\@fileswithoptions#1[#{2}]#3}%
401     {\@fileswithoptions#1[#{2}]#3[]}}
402 \@onlypreamble\@fileswithoptions

```

Then we do some work.

First of all, we define the global variables. Then we look to see if the file has already been loaded. If it has, we check that it was first loaded with at least the current options. If it has not, we add the current options to the package options, set the default version to be 0000/00/00, and load the file if we can find it. Then we check the version number.

Finally, we restore the old file name, reset the default option, and we set the catcode of @.

For classes, we can immediately process the file. For other types, #2 could be a comma separated list, so loop through, processing each one separately.

```

403 </2ekernel>
404 <latexrelease>\IncludeInRelease{2018/12/01}%
405 <latexrelease>       {\@fileswithoptions}{ifx tests in \@fileswithoptions}%
406 <*2ekernel | latexrelease>
407 \def\@fileswithoptions#1[#2]#3[#4]{%
408   \ifx#1\@clsextension
409     \ifx\@classoptionslist\relax
410       \xdef\@classoptionslist{\zap@space#2 \@empty}%

```

Save raw class list.

```

411   \gdef\@raw@classoptionslist{#2}%
412   \def\reserved@a{%
413     \@onefilewithoptions#3[#{2}][#{4}]#1%
414     \@documentclasshook}%
415   \else
416     \def\reserved@a{%
417       \@onefilewithoptions#3[#{2}][#{4}]#1}%
418   \fi
419   \else

```

build up a list of calls to \@onefilewithoptions (one for each package) without thrashing the parameter stack.

```

420   \def\reserved@b##1,{%

```

If #1 is \@nnil we have reached the end of the list (older version used \@nil here but \@nil is undefined so \ifx equal to all undefined commands)

```

421   \ifx\@nnil##1\relax\else

```

If `\ifx\@nnil##1\n@nil` is true then #1 is (presumably) empty (Older code used `\relax` which is slightly easier to get into #1 by mistake, which would spoil this test.)

```

422     \ifx\@nnil##1\@nnil\else
423     \noexpand\@onefilewithoptions##1[#{#2}][#{#4}]%
424     \noexpand\@pkgextension
425     \fi
426     \expandafter\reserved@b
427 \fi}%
428 \edef\reserved@a{\zap@space#3 \@empty}%
429 \edef\reserved@a{\expandafter\reserved@b\reserved@a,\@nnil,}%
430 \fi
431 \reserved@a}
432 </2ekernel | latexrelease)
433 <latexrelease>\EndIncludeInRelease
434 <latexrelease>\IncludeInRelease{2017/01/01}%
435 <latexrelease>    {\@fileswith@pti@ns}{ifx tests in \@fileswith@pti@ns}%
436 <latexrelease>\def\@fileswith@pti@ns#1[#2]#3[#4]{%
437 <latexrelease>  \ifx#1\@clsextension
438 <latexrelease>    \ifx\@classoptionslist\relax
439 <latexrelease>      \xdef\@classoptionslist{\zap@space#2 \@empty}%
440 <latexrelease>      \def\reserved@a{%
441 <latexrelease>        \@onefilewithoptions#3[#{#2}][#{#4}]#1%
442 <latexrelease>        \@documentclasshook}%
443 <latexrelease>    \else
444 <latexrelease>      \def\reserved@a{%
445 <latexrelease>        \@onefilewithoptions#3[#{#2}][#{#4}]#1}%
446 <latexrelease>    \fi
447 <latexrelease>  \else
448 <latexrelease>    \def\reserved@b##1,{%
449 <latexrelease>      \ifx\@nnil##1\relax\else
450 <latexrelease>        \ifx\@nnil##1\@nnil\else
451 <latexrelease>          \noexpand\@onefilewithoptions##1[#{#2}][#{#4}]%
452 <latexrelease>          \noexpand\@pkgextension
453 <latexrelease>          \fi
454 <latexrelease>          \expandafter\reserved@b
455 <latexrelease>          \fi}%
456 <latexrelease>      \edef\reserved@a{\zap@space#3 \@empty}%
457 <latexrelease>      \edef\reserved@a{\expandafter\reserved@b\reserved@a,\@nnil,}%
458 <latexrelease>    \fi
459 <latexrelease>    \reserved@a}
460 <latexrelease>\EndIncludeInRelease
461 <latexrelease>\IncludeInRelease{0000/00/00}%
462 <latexrelease>    {\@fileswith@pti@ns}{ifx tests in \@fileswith@pti@ns}%
463 <latexrelease>\def\@fileswith@pti@ns#1[#2]#3[#4]{%
464 <latexrelease>  \ifx#1\@clsextension
465 <latexrelease>    \ifx\@classoptionslist\relax
466 <latexrelease>      \xdef\@classoptionslist{\zap@space#2 \@empty}%
467 <latexrelease>      \def\reserved@a{%

```

```

468 <latexrelease> \@onefilewithoptions#3[#{#2}][#{#4}]#1%
469 <latexrelease> \@documentclasshook}%
470 <latexrelease> \else
471 <latexrelease> \def\reserved@a{%
472 <latexrelease> \@onefilewithoptions#3[#{#2}][#{#4}]#1}%
473 <latexrelease> \fi
474 <latexrelease> \else
475 <latexrelease> \def\reserved@b##1,{%
476 <latexrelease> \ifx\@nil##1\relax\else
477 <latexrelease> \ifx\relax##1\relax\else
478 <latexrelease> \noexpand\@onefilewithoptions##1[#{#2}][#{#4}]%
479 <latexrelease> \noexpand\@pkgextension
480 <latexrelease> \fi
481 <latexrelease> \expandafter\reserved@b
482 <latexrelease> \fi}%
483 <latexrelease> \edef\reserved@a{\zap@space#3 \@empty}%
484 <latexrelease> \edef\reserved@a{%
485 <latexrelease> \expandafter\reserved@b\reserved@a,\@nil,}%
486 <latexrelease> \fi
487 <latexrelease> \reserved@a}
488 <latexrelease>\EndIncludeInRelease
489 <*2ekernel>

```

```
490 \@onlypreamble\@fileswith@pti@ns
```

Have the main argument as #1, so we only need one \expandafter above.

```

491 \def\@onefilewithoptions#1[#2][#3]#4{%
492 \@pushfilename
493 \xdef\@currname{#1}%
494 \global\let\@currxt#4%
495 \expandafter\let\csname\@currname.\@currxt-h@k\endcsname\@empty
496 \let\CurrentOption\@empty
497 \@reset@options
498 \makeatletter

```

Grab everything in a macro, so the parameter stack is popped before any processing begins.

```

499 \def\reserved@a{%
500 \@ifl@aded\@currxt{#1}%
501 {\@if@options\@currxt{#1}{#2}{}}%
502 {\@latex@error
503 {Option clash for \@cls@pkg\space #1}%
504 {The package #1 has already been loaded
505 with options:\MessageBreak
506 \space\space[\@optionlist{#1.\@currxt}]\MessageBreak
507 There has now been an attempt to load it
508 with options\MessageBreak
509 \space\space[#2]\MessageBreak
510 Adding the global options:\MessageBreak
511 \space\space
512 \@optionlist{#1.\@currxt},#2\MessageBreak

```

```

513         to your \noexpand\documentclass declaration may fix this.%
514         \MessageBreak
515         Try typing \space <return> \space to proceed.}}}%
516     {\@pass@options\@currentx{#2}{#1}%
517     \global\expandafter
518     \let\csname ver@\@currname.\@currentx\endcsname\@empty
519     \InputIfFileExists
520     {\@currname.\@currentx}%
521     {}%
522     {\@missingfileerror\@currname\@currentx}%

```

\@unprocessedoptions will generate an error for each specified option in a package unless a \@ProcessOptions has appeared in the package file.

```

523     \let\@unprocessedoptions\@unprocessedoptions
524     \csname\@currname.\@currentx-h@@k\endcsname
525     \expandafter\let\csname\@currname.\@currentx-h@@k\endcsname
526         \undefined
527     \@unprocessedoptions}%
528     \@ifl@ter\@currentx{#1}{#3}{}%
529     {\@latex@warning@no@line
530     {You have requested,\on@line,
531     version\MessageBreak
532     ‘#3’ of \@cls@pkg\space #1,\MessageBreak
533     but only version\MessageBreak
534     ‘\csname ver@#1.\@currentx\endcsname’\MessageBreak
535     is available}}}%
536     \ifx\@currentx\@clsextension\let\LoadClass\@twoloadclasserror\fi
537     \@popfilename
538     \@reset@options}%
539     \reserved@a}
540 \@onlypreamble\@onefilewithoptions

```

\@fileswith@ptions Save the definition (for error checking).

```

541 \let\@fileswith@ptions\@fileswith@ptions
542 \@onlypreamble\@fileswith@ptions

```

\@reset@options Reset the default option, and clear lists of declared options.

```

543 \def\@reset@options{%
544     \global\ifx\@currentx\@clsextension
545         \let\default@ds\OptionNotUsed
546     \else
547         \let\default@ds\@unknownoptionerror
548     \fi
549     \global\let\ds@\@empty
550     \global\let\@declaredoptions\@empty}
551 \@onlypreamble\@reset@options

```

4.1 Hooks

Allow code do be saved to be executed at specific later times.

Save things in macros, I considered using toks registers, (and `\addto@hook` from the NFSS code, that would require stacking the contents in the case of required packages, so just generate a new macro for each package.

```
\@begindocumenthook Stuff to appear at the beginning or end of the document.
\@enddocumenthook 552 \ifx\@begindocumenthook\@undefined
553 \let\@begindocumenthook\@empty
554 \fi
555 \let\@enddocumenthook\@empty

\g@addto@macro Globally add to the end of a macro.
556 \long\def\g@addto@macro#1#2{%
557 \begingroup
558 \toks@\expandafter{#1#2}%
559 \xdef#1{\the\toks@}%
560 \endgroup}

\AtEndOfPackage The access functions.
\AtEndOfClass 561 \def\AtEndOfPackage{%
\AtBeginDocument 562 \expandafter\g@addto@macro\csname\@currname.\@current-h@k\endcsname}
\AtEndDocument 563 \let\AtEndOfClass\AtEndOfPackage
564 \@onlypreamble\AtEndOfPackage
565 \@onlypreamble\AtEndOfClass

566 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
567 \def\AtEndDocument{\g@addto@macro\@enddocumenthook}
568 \@onlypreamble\AtBeginDocument

\@cls@pkg The current file type.
569 \def\@cls@pkg{%
570 \ifx\@current\@clsextension
571 document class%
572 \else
573 package%
574 \fi}
575 \@onlypreamble\@cls@pkg

\@unknownoptionerror Bad option.
576 \def\@unknownoptionerror{%
577 \@latex@error
578 {Unknown option ‘\CurrentOption’ for \@cls@pkg\space‘\@currname’}%
579 {The option ‘\CurrentOption’ was not declared in
580 \@cls@pkg\space‘\@currname’, perhaps you\MessageBreak
581 misspelled its name.
582 Try typing \space <return>
583 \space to proceed.}}
584 \@onlypreamble\@unknownoptionerror
```

```

\@unprocessedoptions Declare an error for each option, unless a \ProcessOptions occurred.
585 \def\@unprocessedoptions{%
586   \ifx\@current\@pkgextension
587     \edef\@curroptions{\@optionlist{\@currname.\@current}}}%
588     \for\CurrentOption:=\@curroptions\do{%
589       \ifx\CurrentOption\@empty\else\@unknownoptionerror\fi}%
590   \fi}
591 \@onlypreamble\@unprocessedoptions
592 \@onlypreamble\@unprocessedoptions

\@badrequireerror \RequirePackage or \LoadClass occurs in the options section.
593 \def\@badrequireerror#1[#2]#3[#4]{%
594   \@latex@error
595     {Two \noexpand\RequirePackage or \noexpand\LoadClass
596       in Options Section}%
597     {The \@cls@pkg\space ‘\@currname’ is defective.\MessageBreak
598       It attempts to load ‘#3’ in the options section, i.e.,\MessageBreak
599       between \noexpand\DeclareOption and \string\ProcessOptions.}}
600 \@onlypreamble\@badrequireerror

\@twoloadclasserror Two \LoadClass in a class.
601 \def\@twoloadclasserror{%
602   \@latex@error
603     {Two \noexpand\LoadClass commands}%
604     {You may only use one \noexpand\LoadClass in a class file}}
605 \@onlypreamble\@twoloadclasserror

\@twoclasseserror Two \documentclass or \documentstyle.
606 \def\@twoclasseserror#1#2{%
607   \@latex@error
608     {Two \noexpand\documentclass or \noexpand\documentstyle commands}%
609     {The document may only declare one class.}\@gobble}
610 \@onlypreamble\@twoclasseserror

4.2 Providing shipment

\two@digits Prefix a number less than 10 with ‘0’.
611 \def\two@digits#1{\ifnum#1<10 0\fi\number#1}

\filecontents This environment implements inline files. The star-form does not write extra
\endfilecontents comments into the file.
612 \begingroup%
613 \@tempcnta=1
614 \loop
615   \catcode\@tempcnta=12 %
616   \advance\@tempcnta\@ne %
617   \ifnum\@tempcnta<32 %
618   \repeat %

```

```

619 \catcode\*=11 %
620 \catcode\^^M\active%
621 \catcode\^^L\active\let^^L\relax%
622 \catcode\^^I\active%

623 \gdef\filecontents{\@tempswatruel\filec@ntents}%
624 \gdef\filecontents*{\@tempswafalse\filec@ntents}%

625 \gdef\filec@ntents#1{%
626   \openin\@inputcheck#1 %
627   \ifeof\@inputcheck%
628     \@latex@warning@no@line%
629     {Writing file '\@currdir#1'}%

630   \chardef\reserved@c15 %
631   \ch@ck7\reserved@c\write%
632   \immediate\openout\reserved@c#1\relax%
633   \else%

634     \closein\@inputcheck%
635     \@latex@warning@no@line%
636     {File '#1' already exists on the system.\MessageBreak%
637     Not generating it from this source}%
638     \let\write\@gobbletwo%
639     \let\closeout\@gobble%
640   \fi%
641   \if@tempswa%

642     \immediate\write\reserved@c{%
643       \@percentchar\@percentchar\space%
644       \expandafter\@gobble\string\LaTeX2e file '#1'^^J%
645       \@percentchar\@percentchar\space generated by the %
646       '\@currvir' \expandafter\@gobblefour\stringnewenvironment^^J%
647       \@percentchar\@percentchar\space from source '\jobname' on %
648       \number\year/\two@digits\month/\two@digits\day.^^J%
649       \@percentchar\@percentchar}%
650   \fi%
651   \let\do\@makeother\dospecials%

If there are active characters in the upper half (e.g., from inputenc there would
be confusion so we render everything harmless.

652 \count@ 128\relax%
653 \loop%
654   \catcode\count@ 11\relax%
655   \advance\count@ \@ne%
656   \ifnum\count@<\@cclvi%
657   \repeat%

658 \edef\E{\@backslashchar end\string{\@currvir\string}}%
659 \edef\reserved@b{%
660   \def\noexpand\reserved@b%
661     #####1\E####2\E####3\relax}%
662 \reserved@b{%
663   \ifx\relax##3\relax%

```

```

There was no \end{filecontents}
664     \immediate\write\reserved@c{##1}%
665     \else%
There was a \end{filecontents}, so stop this time.
666     \edef^^M{\noexpand\end{\@currentenv}}%
667     \ifx\relax##1\relax%
668     \else%
Text before the \end, write it with a warning.
669         \@latex@warning{Writing text '##1' before %
670             \string\end{\@currentenv}\MessageBreak as last line of #1}%
671         \immediate\write\reserved@c{##1}%
672         \fi%
673         \ifx\relax##2\relax%
674         \else%
Text after the \end, ignore it with a warning.
675         \@latex@warning{%
676             Ignoring text '##2' after \string\end{\@currentenv}}%
677         \fi%
678     \fi%
679     ^^M}%

680 \catcode'\^^L\active%
681 \let\L@undefined%
682 \def^^L{\expandafter\ifx\csname L\endcsname\relax\fi ^^J^^J}%
683 \catcode'\^^I\active%
684 \let\I@undefined%
685 \def^^I{\expandafter\ifx\csname I\endcsname\relax\fi\space}%
686 \catcode'\^^M\active%
687 \edef^^M##1^^M{%
688     \noexpand\reserved@b##1\E\E\relax}}%
689 \endgroup

690 \begingroup
691 \catcode'\|=\catcode'\%
692 \catcode'\%=12
693 \catcode'\*=11
694 \gdef\@percentchar{%}
695 \gdef\endfilecontents{|
696     \immediate\closeout\reserved@c
697     \def\T##1##2##3{|
698     \ifx##1\@undefined\else
699         \@latex@warning@no@line{##2 has been converted to Blank ##3e}|
700     \fi}|
701 \T\L{Form Feed}{Lin}|
702 \T\I{Tab}{Spac}|
703 \immediate\write\@unused{}}
704 \global\let\endfilecontents*\endfilecontents
705 \@onlypreamble\filecontents
706 \@onlypreamble\endfilecontents

```



```

707 \@onlypreamble\filecontents*
708 \@onlypreamble\endfilecontents*
709 \endgroup
710 \@onlypreamble\filecontents

```

5 Package/class rollback mechanism

```

711 </2ekernel>
712 <*2ekernel | latexreleasefirst>

```

`\pkgcls@debug` For testing we have a few extra lines of code that by default do nothing but one can set `\pkgcls@debug` to `\typeout` to get extra info. Sometime in the future this will be dropped.

```

713 <*tracerollback>
714 %\let \pkgcls@debug\typeout
715 \let \pkgcls@debug@gobble
716 </tracerollback>

```

`\requestedLaTeXdate` The macro (!) `\requestedLaTeXdate` holds the globally requested rollback date (via `latexrelease`) or zero if no such request was made.

```

717 \def \requestedLaTeXdate{0}

```

`\pkgcls@targetdate` If a rollback for a package or class is requested then `\pkgcls@targetdate` holds the requested date as a number YYYYMMDD (if there was one, otherwise the value of `\requestedLaTeXdate`) and `\pkgcls@targetlabel` will be empty. If there was a request for a named version then `\pkgcls@targetlabel` holds the version name and `\pkgcls@targetdate` is set to 1.

`\pkgcls@targetdate=0` is used to indicate that there was no rollback request. While loading an old release `\pkgcls@targetdate` is also reset to zero so that `\DeclareRelease` declarations are bypassed.

In contrast `\pkgcls@innerdate` will always hold the requested date (in a macro not a counter) if there was one, otherwise, e.g., if there was no request or a request to a version name it will contain T_EX largest legal number. While loading a file this can be used to provide conditionals that select code based on the request.

```

718 \ifx \pkgcls@targetdate \@undefined
719   \newcount \pkgcls@targetdate
720 \fi
721 \let \pkgcls@targetlabel \@empty
722 \def \pkgcls@innerdate {\maxdimen}

```

`\pkgcls@candidate` When looping through the `\DeclareRelease` declarations we record if the release is the best candidate we have seen so far. This is recorded in `\pkgcls@candidate` and we update it whenever we see a better one.

In `\pkgcls@releasedate` we keep track of the release date of that candidate.

```

723 \let \pkgcls@candidate \@empty
724 \let \pkgcls@releasedate \@empty

```

`\load@onefilewithoptions` the best place to add the rollback code is at the point where `\@onefilewithoptions` is called to load a single class or package.

To make things easy we save the old definition as `\load@onefilewithoptions` and then provide a new interface.

Important: as this code is also unconditionally placed into latexrelease we can only do this name change once otherwise both macros will contain the same code.

```
725 \ifx\load@onefilewithoptions\@undefined
726 \let\load@onefilewithoptions\@onefilewithoptions
727 \def\@onefilewithoptions#1[#2][#3]#4{%
```

First a bit of tracing normally disabled.

```
728 <*tracerollback>
729 \pkgcls@debug{--- File loaded request (\noexpand\usepackage or ...)}%
730 \pkgcls@debug{\@spaces 1: #1}%
731 \pkgcls@debug{\@spaces 2: #2}%
732 \pkgcls@debug{\@spaces 3: #3}%
733 \pkgcls@debug{\@spaces 4: #4}%
734 </tracerollback>
```

Two of the arguments are needed later on in error/warning messages so we save them.

```
735 \def\pkgcls@name{#1}%           % for info message
736 \def\pkgcls@arg {#3}%          % for info message
```

then we parse the final optional argument to determine if there is a specific rollback request for the current file. This will set `\pkgcls@targetdate`, `\pkgcls@targetlabel` and `\pkgcls@mindate`.

```
737 \pkgcls@parse@date@arg{#3}%
```

When determining the correct release to load we keep track of candidates in `\pkgcls@candidate` and initially we don't have any:

```
738 \let\pkgcls@candidate\@empty
```

If we had a rollback request then #3 may contain data but not necessarily a “minimal date” so instead of passing it on we pass on the content of `\pkgcls@mindate`. We need to pass the value not the command, otherwise nested packages may pick up the wrong information.

```
739 \begingroup
740 \edef\reserved@a{%
741   \endgroup
742   \unexpanded{\load@onefilewithoptions#1[#2]}%
743   [\pkgcls@mindate]%
744   \unexpanded{#4}}%
745 \reserved@a
746 }
747 \fi
```

`\pkgcls@parse@date@arg` The `\pkgcls@parse@date@arg` command parses the second optional argument of `\usepackage`, `\RequirePackage` or `\documentclass` for a rollback request setting the values of `\pkgcls@targetdate` and `\pkgcls@targetlabel`.

This optional argument has a dual purpose: If it just contains a date string then this means that the package should have at least that date (to ensure that a certain feature is actually available, or a certain bug has been fixed). When the package gets loaded the information in `\Provides...` will then be checked against this request.

But if it starts with an equal sign followed by a date string or followed by a version name then this means that we should roll back to the state of the package at that date or to the version with the requested name.

If there was no optional argument or the optional argument does not start with “=” then the `\pkgcls@targetdate` is set to the date of the overall rollback request (via `latexrelease`) or if that was not given it is set to 0. In either case `\pkgcls@targetlabel` will be made empty.

If the argument doesn’t start with “=” then it is supposed to be a “minimal date” and we therefore save the value in `\pkgcls@mindate`, otherwise this macro is made empty.

So in summary we have:

Input	<code>\pkgcls@targetdate</code>	<code>\pkgcls@targetlabel</code>	<code>\pkgcls@mindate</code>
<code><empty></code>	→ <code><global-rollbackdate-as-number></code>	<code><empty></code>	<code><empty></code>
<code><date></code>	→ <code><global-rollbackdate-as-number></code>	<code><empty></code>	<code><date></code>
<code>=<date></code>	→ <code><date-as-number></code>	<code><empty></code>	<code><empty></code>
<code>=<version></code>	→ <code>1</code>	<code><version></code>	<code><empty></code>
<code><other></code>	→ <code><global-rollbackdate-as-number></code>	<code><empty></code>	<code><other></code>

where `<global-rollbackdate-as-number>` is a date request given via `latexrelease` or if there wasn’t one 0.

```
748 \def\pkgcls@parse@date@arg #1{%
```

If the argument is empty we use the rollback date from `latexrelease` which has the value of zero if there was no rollback request. The label and the minimal date is made empty in that case.

```
749 \ifx\@nil#1\@nil
750 \pkgcls@targetdate\requestedLaTeXdate\relax
751 \let\pkgcls@targetlabel\@empty
752 \let\pkgcls@mindate\@empty
```

Otherwise we parse the argument further, checking for a = as the first character. We append a = at the end so that there is at least one such character in the argument.

```
753 \else
754 \pkgcls@parse@date@arg@#1=\@nil\relax
755 \fi
756 }
```

The actual parsing work then happens in `\pkgcls@parse@date@arg@`:

```
757 \def\pkgcls@parse@date@arg@#1=#2\@nil{%
```

We set `\pkgcls@targetdate` depending on the parsing result; the code is expandable so we can do the parsing as part of the assignment.

```
758 \pkgcls@targetdate
```

If a = was in first position then #1 will be empty. In that case #2 will be the original argument with a = appended.

This can be parsed with \@parse@version, the trailing character is simply ignored. This macro returns the parsed date as a number (or zero if it wasn't a date) and accepts both YYYY/MM/DD and YYYY-MM-DD formats.

```
759 \ifx\@nil#1\@nil
760 \@parse@version0#2//00\@nil\relax
```

Whatever is returned is thus assigned to \pkgcls@targetdate and therefore we can now test its value. If the value is zero we assume that the remaining argument string represents a version and change \pkgcls@targetdate and set \pkgcls@targetlabel to the version name (after stripping off the trailing =.

```
761 \ifnum \pkgcls@targetdate=\z@
762 \pkgcls@targetdate\@ne
763 \def\pkgcls@innerdate{\maxdimen}%
764 \pkgcls@parse@date@arg@version#2%
765 \else
766 \edef\pkgcls@innerdate{\the\pkgcls@targetdate}%
767 \fi
768 \let\pkgcls@mindate\@empty
769 \else
```

If #1 was not empty then there wasn't a = character in first position so we are dealing either with a "minimum date" or with some incorrect data. We assume the former and make the following assignments (the first one finishing the assignment of \pkgcls@targetdate):

```
770 \requestedLaTeXdate\relax
771 \let\pkgcls@targetlabel\@empty
772 \def\pkgcls@innerdate{\maxdimen}%
773 \def\pkgcls@mindate{#1}%
```

If the min-date is after the requested rollback date (if there is any, i.e., if it is not zero) then we have a conflict and therefore issue a warning.

```
774 \ifnum \pkgcls@targetdate > \z@
775 \ifnum \@parse@version0#1//00\@nil > \pkgcls@targetdate
776 \@latex@warning@no@line{Suspicious rollback/min-date date given\MessageBreak
777 A minimal date of #1 has been specified for
778 \@cls@pkg\MessageBreak '\pkgcls@name'.\MessageBreak
779 But this is in conflict
780 with a rollback request to \requestedpatchdate}
781 \fi
782 \fi
783 \fi
784 }
```

Strip off the trailing = and assign the version name to \pkgcls@targetlabel.

```
785 \def\pkgcls@parse@date@arg@version#1={%
786 \def\pkgcls@targetlabel{#1}}
```

`\DeclareRelease` First argument is the “name” of the release and it can be left empty if one doesn’t like to give a name to the release. The second argument is that from which on this release was available (or should be used in case of minor updates). The final argument is the external file name of this release, by convention this should be `<pkg/cls-name>-<date>.<extension>` but this is not enforced and through this argument one can overwrite it.

```

787 \def\DeclareRelease#1#2#3{%
788   \ifnum\pkgcls@targetdate>\z@ % some sort of rollback request
789   \traceroollback
790     \pkgcls@debug{---\string\DeclareRelease:}%
791     \pkgcls@debug{\@spaces 1: #1}%
792     \pkgcls@debug{\@spaces 2: #2}%
793     \pkgcls@debug{\@spaces 3: #3}%
794 \traceroollback}

```

If the date argument #2 is empty we are dealing with a special release that should be only accessible via its name; a typical use case would be a “beta” release. So if we are currently processing a date request we ignore it and otherwise we check if we can match the name and if so load the corresponding release file.

```

795   \ifx\@nil#2\@nil
796     \ifnum\pkgcls@targetdate=\@ne % named request
797       \def\reserved@a{#1}%
798       \ifx\pkgcls@targetlabel\reserved@a
799         \pkgcls@use@this@release{#3}{}%
800 \traceroollback}
801   \else
802     \pkgcls@debug{Label doesn't match}%
803 \traceroollback}
804   \fi
805 \traceroollback}
806 \else
807   \pkgcls@debug{Date request: ignored}%
808 \traceroollback}
809 \fi
810 \else

```

If the value of `\pkgcls@targetdate` is greater than 1 (or in reality greater than something like 19930101) we are dealing with a rollback request to a specific date.

```

811   \ifnum\pkgcls@targetdate>\@ne % a real request

```

So we parse the date of this release to check if it is before or after the request date.

```

812     \ifnum\@parse@version#2//00\@nil
813       >\pkgcls@targetdate

```

If it is after we have to distinguish between two cases: If there was an earlier candidate we use that one because the other is too late, but if there wasn’t one (i.e., if current release is the oldest that exists) we use it as the best choice. However in that case something is wrong (as there shouldn’t be a rollback to a date where a package used doesn’t yet exists. So we make a complained to the user.

```

814         \ifx\pkgcls@candidate\@empty
815             \pkgcls@rollbackdate@error{#2}%
816             \pkgcls@use@this@release{#3}{#2}%
817         \else
818             \pkgcls@use@this@release\pkgcls@candidate
819                                     \pkgcls@releasedate
820         \fi
821     \else

```

Otherwise, if the release date of this version is before the target rollback and we record it as a candidate. But we don't use it yet as there may be another release which is still before the target rollback.

```

822         \def\pkgcls@candidate{#3}%
823         \def\pkgcls@releasedate{#2}%
824     \tracerelease
825         \pkgcls@debug{New candidate: #3}%
826     \tracerelease
827     \fi
828 \else

```

If we end up in this branch we have a named version request. So we check if `\pkgcls@targetlabel` matches the current name and if yes we use this release immediately, otherwise we do nothing as a later declaration may match it.

```

829         \def\reserved@a{#1}%
830         \ifx\pkgcls@targetlabel\reserved@a
831             \pkgcls@use@this@release{#3}{#2}%
832     \tracerelease
833     \else
834         \pkgcls@debug{Label doesn't match}%
835     \tracerelease
836     \fi
837 \fi
838 \fi
839 \fi
840 }

```

`\pkgcls@use@this@release` If a certain release has been selected (stored in the external file given in #1) we need to input it and afterwards stop reading the current file.

```

841 \def\pkgcls@use@this@release#1#2{%

```

Before that we record the selection made inside the transcript.

```

842 \pkgcls@show@selection{#1}{#2}%

```

We then set the `\pkgcls@targetdate` to zero so that any `\DeclareRelease` or `\DeclareCurrentRelease` in the file we now load are bypassed¹ and then we finally load the correct release.

After loading that file we need to stop reading the current file so we issue `\endinput`. Note that the `\relax` before that is essential to ensure that the

¹The older release may also have such declarations inside if it was a simply copy of the `.sty` or `.cls` file current at that date. Removing these declarations would make the file load a tiny bit faster, but this way it works in any case.

`\endinput` is only happening after the file has been fully processed, otherwise it would act after the first line of the `\@@input!`

```
843 \pkgcls@targetdate\z@
844 \@@input #1\relax
845 \endinput
846 }
```

`\pkgcls@show@selection` This command records what selection was made. As that is needed in two places (and it is rather lengthy) it was placed in a separate command. The first argument is the name of the external file that is being loaded and is only needed for debugging. The second argument is the date that corresponds to this file and it is used as part of the message.

```
847 \def\pkgcls@show@selection#1#2{%
848 \*traceroollback)
849 \pkgcls@debug{Result: use #1}%
850 \traceroollback)
851 \GenericInfo
852 {\@spaces\@spaces\space}{Rollback for
853 \@cls@pkg\space'\@currname' requested ->
854 \ifnum\pkgcls@targetdate>\@ne
855 date
856 \ifnum\requestedLaTeXdate=\pkgcls@targetdate
857 \requestedpatchdate
858 \else
859 \expandafter\@gobble\pkgcls@arg
860 \fi.\MessageBreak
```

Instead of “best approximation” we could say that we have been able to exactly match the date (if it is exact), but that would mean extra tests without much gain, so not done.

```
861 Best approximation is
862 \else
863 version '\pkgcls@targetlabel'.\MessageBreak
864 This corresponds to
865 \fi
866 \ifx\@nil#2\@nil
867 a special release%
868 \else
869 the release introduced on #2%
870 \fi
871 \@gobble}%
872 }
```

`\pkgcls@rollbackdate@error` This is called if the requested rollback date is earlier than the earliest known release of a package or class.

A similar error is given if global rollback date and min-date on a specific package conflict with each other, but that case is happens only once so it is inlined.

```
873 \def\pkgcls@rollbackdate@error#1{%
```

```

874 \latex@error{Suspicious rollback date given}%
875 {The \cls@pkg\space'\currname' claims that it
876 came into existence on #1 which\MessageBreak
877 is after your requested rollback date --- so
878 something is wrong here.\MessageBreak
879 Continue and we use the earliest known release.}}

```

`\DeclareCurrentRelease` This declares the date (and possible name) of the current version of a package or class.

```

880 \def\DeclareCurrentRelease#1#2{%
      First we test if \pkgcls@targetdate is greater than zero, otherwise this code is
      bypassed (as there is no rollback request).
881 \ifnum\pkgcls@targetdate>\z0 % some sort of rollback request
882 (*tracereollback)
883 \pkgcls@debug{---DeclareCurrentRelease}%
884 \pkgcls@debug{ 1: #1}%
885 \pkgcls@debug{ 2: #2}%
886 (/tracereollback)

```

If the value is greater than 1 we have to deal with a date request, so we parse #2 as a date and compare it with `\pkgcls@targetdate`.

```

887 \ifnum\pkgcls@targetdate>\@ne % a date request
888 \ifnum\@parse@version#2//00\@nil
889 >\pkgcls@targetdate

```

If it is greater that means the release date if this file is later than the requested rollback date. Again we have two cases: If there was a previous candidate release we use that one as the current release is too young, but if there wasn't we have to use this release nevertheless as there isn't any alternative.

However this case can only happen if there is a `\DeclareCurrentRelease` but no declared older releases (so basically the use of the declaration is a bit dubious).

```

890
891 \ifx\pkgcls@candidate\@empty
892 \pkgcls@rollbackdate@error{#2}%
893 \else
894 \pkgcls@use@this@release\pkgcls@candidate
895 \pkgcls@releasedate
896 \fi

```

Otherwise the current file is the right release, so we record that in the transcript and then carry on.

```

897 \else
898 \pkgcls@show@selection{current version}{#2}%
899 \fi
900 \else % a label request

```

Otherwise we have a rollback request to a named version so we check if that fits the current name and if not give an error as this was the last possible opportunity.

```

901 \def\reserved@a{#1}%
902 \ifx\pkgcls@targetlabel\reserved@a

```



```

903     \pkgcls@show@selection{current version}{#2}%
904     \else
905     \latex@error{Requested version '\pkgcls@targetlabel' for
906     \cls@pkg\space'\currname' is unknown}\@ehc
907     \fi
908     \fi
909     \fi
910 }

```

`\IfTargetDateBefore` This enables a simple form of conditional code inside a class or package file. If there is a date request and the request date is earlier than the first argument the code in the second argument is processed otherwise the code in the third argument is processed. If there was no date request then we also execute the third argument, i.e., we will get the “latest” version of the file.

Most often the second argument (before-date-code) will be empty.

```

911 \long\def\IfTargetDateBefore#1{%
912   \ifnum\pkgcls@innerdate <%
913     \expandafter\@parse@version\expandafter0#1//00\@nil
914     \typeout{Exclude code introduced on #1}%
915     \expandafter\@firstoftwo
916   \else
917     \typeout{Include code introduced on #1}%
918     \expandafter\@secondoftwo
919   \fi
920 }

```

```

921 </2ekernel | latexreleasefirst>

```

6 After Preamble

Finally we declare a package that allows all the commands declared above to be `\@onlypreamble` to be used after `\begin{document}`.

```

922 <*afterpreamble>
923 \NeedsTeXFormat{LaTeX2e}
924 \ProvidesPackage{pkgindoc}
925     [1994/10/20 v1.1 Package Interface in Document (DPC)]
926 \def\reserved@a#1\do\@classoptionslist#2\do\filecontents#3\relax{%
927   \gdef\@preamblecmds{#1#3}}
928 \expandafter\reserved@a\@preamblecmds\relax
929 </afterpreamble>

```