

The main structure of documents

Frank Mittelbach Chris Rowley Alan Jeffrey
David Carlisle

2018/03/27

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
\@ifclassloaded
\@ifpackagelater
\@ifclasslater
\@ifpackagewith
\@ifclasswith
```

To find out if a package has already been loaded, use

```
\@ifpackageloaded{<package>}{<true>}{<false>}.
```

To find out if a package has already been loaded with a version equal to or more recent than `<version>`, use

```
\@ifpackagelater{<package>}{<version>}{<true>}{<false>}.
```

To find out if a package has already been loaded with at least the options `<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
\DeclareOption*
```

To define a builtin option, use `\DeclareOption{<name>}{<code>}`.

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*{\@unknownoptionerror}
```

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

<code>\InputIfFileExists</code>	<code>\InputIfFileExists{<file>}{<then>}{<else>}</code> Inputs <i><file></i> if it exists. Immediately before the input, <i><then></i> is executed. Otherwise <i><else></i> is executed.
<code>\IfFileExists</code>	As above, but does not input the file. One thing you might like to put in the <i><else></i> clause is
<code>\@missingfileerror</code>	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,
<code>\input</code>	This has been redefined from the L ^A T _E X2.09 definition, in terms of the new commands <code>\InputIfFileExists</code> and <code>\@missingfileerror</code> .
<code>\listfiles</code>	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 <code>\ProvidesPackage</code> 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 (*2ekernel)

\if@compatibility The flag for compatibility mode.
2 \newif\if@compatibility

\@documentclasshook The hook called after the first \documentclass command. By default this checks
to see if \@normalsize is undefined, and if so, sets it to \normalsize.
3 \def\@documentclasshook{%
4   \ifx\@normalsize\undefined
5     \let\@normalsize\normalsize
6   \fi
7 }

\@declaredoptions This list is automatically built by \DeclareOption. 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 \ds<option> commands are executed. All local <option>s which are not declared will be processed in the order defined by the optional argument of \documentclass or \usepackage.
8 \let\@declaredoptions\@empty

\@classoptionslist List of options of the main class.
9 \let\@classoptionslist\relax
10 \onlypreamble\@classoptionslist

```

<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>\@currentx</code>	The current file extension. 15 <code>\global\let\@currentx=\@empty</code>
<code>\@clsextension</code>	The two possible values of <code>\@currentx</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>
<code>\@pushfilename</code>	Commands to push and pop the file name and extension.
<code>\@popfilename</code>	#1 current name.
<code>\@currnamestack</code>	#2 current extension. #3 current catcode of @. #4 Rest of the stack. 20 <code>\def\@pushfilename{%</code> 21 <code> \edef\@currnamestack{%</code> 22 <code> {\@currname}%</code> 23 <code> {\@currentx}%</code> 24 <code> {\the\catcode`\@}%</code> 25 <code> \@currnamestack}}</code> 26 <code>\@onlypreamble\@pushfilename</code> 27 <code>\def\@popfilename{\expandafter\@p@pfilename\@currnamestack\@nil}</code> 28 <code>\@onlypreamble\@popfilename</code> 29 <code>\def\@p@pfilename#1#2#3#4\@nil{%</code> 30 <code> \gdef\@currname{#1}%</code> 31 <code> \gdef\@currentx{#2}%</code> 32 <code> \catcode`\@#3\relax</code> 33 <code> \gdef\@currnamestack{#4}}</code> 34 <code>\@onlypreamble\@p@pfilename</code> 35 <code>\gdef\@currnamestack{}</code> 36 <code>\@onlypreamble\@currnamestack</code>
<code>\@optionlist</code>	Returns the option list of the file. 37 <code>\def\@optionlist#1{%</code> 38 <code> \ifundefined{opt@#1}\@empty{\csname opt@#1\endcsname}}</code> 39 <code>\@onlypreamble\@optionlist</code>
<code>\@ifpackageloaded</code>	<code>\@ifpackageloaded{<name>}</code> Checks to see whether a file has been loaded.
<code>\@ifclassloaded</code>	40 <code>\def\@ifpackageloaded{\@ifl@ded\@pkgextension}</code> 41 <code>\def\@ifclassloaded{\@ifl@ded\@clsextension}</code> 42 <code>\@onlypreamble\@ifpackageloaded</code> 43 <code>\@onlypreamble\@ifclassloaded</code>

```

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 more recent than the given date.

51 \def\@ifpackagelater{\@ifl@ter\@pkgextension}
52 \def\@ifclasslater{\@ifl@ter\@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\expandafter0#1//00\@nil<%
65     \expandafter\@parse@version\expandafter0#2//00\@nil
66   \expandafter\@secondoftwo
67   \else
68     \expandafter\@firstoftwo
69   \fi}
70 \</2ekernel | latexrelease>
71 \<latexrelease>\EndIncludeInRelease
72 \<latexrelease>\IncludeInRelease{0000/00/00}%
73 \<latexrelease> \<@ifl@t@r>\{Guard against bad input}%
74 \<latexrelease>\def\@ifl@t@r#1#2{%
75 \<latexrelease> \ifnum\expandafter\@parse@version#1//00\@nil<%
76 \<latexrelease> \expandafter\@parse@version#2//00\@nil
77 \<latexrelease> \expandafter\@secondoftwo
78 \<latexrelease> \else
79 \<latexrelease> \expandafter\@firstoftwo
80 \<latexrelease> \fi}
81 \<latexrelease>\EndIncludeInRelease
82 \<*2ekernel>

83 \@onlypreamble\@ifl@t@r
84 \</2ekernel>
85 \<*2ekernel | latexreleasefirst>
86 \def\@parse@version#1/#2/#3#4#5\@nil{%
87 \@parse@version@dash#1-#2-#3#4\@nil
88 }

```

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

```

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

\@ifpackagewith \<@ifpackagewith{<name>}<{<option-list>}> Checks that <option-list> is a subset of
\@ifclasswith the options with which <name> was loaded.
93 \def\@ifpackagewith{\@ifoptions\@pkgextension}
94 \def\@ifclasswith{\@ifoptions\@clsextension}
95 \<onlypreamble\@ifpackagewith
96 \<onlypreamble\@ifclasswith

97 \def\@ifoptions#1#2{%
98   \@expandtwoargs\@if@pti@ns{\@optionlist{#2.#1}}
99 \<onlypreamble\@ifoptions

Probably shouldn't use \CurrentOption here...(changed to \reserved@b.)
100 \</2ekernel>
101 \<latexrelease>\IncludeInRelease{2017/01/01}%
102 \<latexrelease>           {\@if@pti@ns}{Spaces in option clash check}%
103 \<2ekernel|latexrelease>
104 \def\@if@pti@ns#1#2{%
105   \let\reserved@a\@firstoftwo

106   \edef\reserved@b{\zap@space#2 \@empty}%
107   \@for\reserved@b:=\reserved@b\do{%
108     \ifx\reserved@b\@empty
109       \else
110         \expandafter\in@\expandafter{\expandafter,\reserved@b,}{, #1,}%
111         \ifin@
112         \else
113           \let\reserved@a\@secondoftwo
114         \fi
115       \fi
116     }%
117   \reserved@a}
118 \</2ekernel|latexrelease>
119 \<latexrelease>\EndIncludeInRelease
120 \<latexrelease>\IncludeInRelease{0000/00/00}%
121 \<latexrelease>           {\@if@pti@ns}{Spaces in option clash check}%
122 \<latexrelease>\def\@if@pti@ns#1#2{%
123 \<latexrelease> \let\reserved@a\@firstoftwo
124 \<latexrelease> \@for\reserved@b:=#2\do{%
125 \<latexrelease> \ifx\reserved@b\@empty
126 \<latexrelease> \else
127 \<latexrelease> \expandafter\in@\expandafter
128 \<latexrelease>           {\expandafter,\reserved@b,}{, #1,}%
129 \<latexrelease> \ifin@
130 \<latexrelease> \else
131 \<latexrelease> \let\reserved@a\@secondoftwo
132 \<latexrelease> \fi
133 \<latexrelease> \fi
134 \<latexrelease> }%
135 \<latexrelease> \reserved@a}
136 \<latexrelease>\EndIncludeInRelease
137 \<2ekernel>

```

```

138 \@onlypreamble\@ifopti@ns

\ProvidesPackage Checks that the current filename is correct, and defines \ver@filename.
139 \def\ProvidesPackage#1{%
140   \xdef\@gtempa{#1}%
141   \ifx\@gtempa\@currname\else
142     \@latex@warning@no@line{You have requested
143       \@cls@pkg\space` \@currname', \MessageBreak
144       but the \@cls@pkg\space provides `#1'}%
145   \fi
146   \@ifnextchar[\@pr@videpackage{\@pr@videpackage[]}]
147 \@onlypreamble\ProvidesPackage

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

\ProvidesClass Like \ProvidesPackage, but for classes.
156 \let\ProvidesClass\ProvidesPackage
157 \@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
158 \def\ProvidesFile#1{%
159   \begingroup
160     \catcode\ 10 %
161     \ifnum \endlinechar<256 %
162       \ifnum \endlinechar>\m@ne
163         \catcode\endlinechar 10 %
164       \fi
165     \fi
166     \@makeother\/%
167     \@makeother\&%
168     \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 ltfinal.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.
169 \def\@pass@options#1#2#3{%
170   \expandafter\xdef\csname opt@#3.#1\endcsname{%
171     \ifundefined{opt@#3.#1}\@empty

```

```

172      {\csname opt@#3.#1\endcsname,}%
173      \zap@space#2 \@empty}}
174 \@onlypreamble\@pass@options

175 \def\PassOptionsToPackage{\@pass@options\@pkgextension}
176 \def\PassOptionsToClass{\@pass@options\@clsextension}
177 \@onlypreamble\PassOptionsToPackage
178 \@onlypreamble\PassOptionsToClass

\DeclareOption Adds an option as a \ds@ command, or the default \default@ds command.
\DeclareOption*
179 \def\DeclareOption{%
180   \let\@fileswith@pti@ns\@badrequireerror
181   \ifstar\@defdefault@ds\@declareoption}
182 \long\def\@declareoption#1#2{%
183   \xdef\@declaredoptions{\@declaredoptions,#1}%
184   \toks@{#2}%
185   \expandafter\edef\csname ds@#1\endcsname{\the\toks@}}
186 \long\def\@defdefault@ds#1{%
187   \toks@{#1}%
188   \edef\default@ds{\the\toks@}}
189 \@onlypreamble\DeclareOption
190 \@onlypreamble\@declareoption
191 \@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.
192 \def\OptionNotUsed{%
193   \ifx\@current\@clsextension
194     \xdef\@unusedoptionlist{%
195       \ifx\@unusedoptionlist\@empty\else\@unusedoptionlist,\fi
196       \CurrentOption}%
197   \fi}
198 \@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*.
199 % \let\default@ds\OptionNotUsed

\ProcessOptions \ProcessOptions calls \ds@option for each known package option, then calls
\ProcessOptions* \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.

200 \def\ProcessOptions{%
201   \let\ds@\@empty
202   \edef\@curroptions{\@optionlist{\@currname.\@current}}%
203   \@ifstar\@xprocess@options\@process@ptions}
204 \@onlypreamble\ProcessOptions

```

```

205 \def\@process@ptions{%
206   \for\CurrentOption:=\@declaredoptions\do{%
207     \ifx\CurrentOption\@empty\else
208       \expandtwoargs\in{,\CurrentOption,}{%
209         ,\ifx\@current\@clsextension\else\@classoptionslist,\fi
210         \@curroptions,}%
211     \ifin@
212       \@use@option
213       \expandafter\let\csname ds@\CurrentOption\endcsname\@empty
214     \fi
215   \fi}%
216 \@process@ptions}
217 \@onlypreamble\@process@ptions

218 \def\@xprocess@ptions{%
219   \ifx\@current\@clsextension\else
220     \for\CurrentOption:=\@classoptionslist\do{%
221       \ifx\CurrentOption\@empty\else
222         \expandtwoargs\in{,\CurrentOption,}{,\@declaredoptions,}%
223       \ifin@
224         \@use@option
225         \expandafter\let\csname ds@\CurrentOption\endcsname\@empty
226       \fi
227     \fi}%
228 \fi
229 \@process@ptions}
230 \@onlypreamble\@xprocess@ptions

```

The common part of \ProcessOptions and \ProcessOptions*.

```

231 \def\@process@ptions{%
232   \for\CurrentOption:=\@curroptions\do{%
233     \@ifundefined{ds@\CurrentOption}%
234     {\@use@option
235      \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.

```

236     \@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.

```

237   \for\CurrentOption:=\@declaredoptions\do{%
238     \expandafter\let\csname ds@\CurrentOption\endcsname\relax}%
239   \let\CurrentOption\@empty
240   \let\@fileswith@ptions\@fileswith@ptions
241   \AtEndOfPackage{\let\@unprocessedoptions\relax}}
242 \@onlypreamble\@process@ptions

```

\@options \@options is a synonym for \ProcessOptions* for upward compatibility with L^AT_EX2.09 style files.

```

243 \def\@options{\ProcessOptions*}
244 \@onlypreamble\@options

```

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

```

245 \def\@use@option{%
246   \@expandtwoargs\@removeelement\CurrentOption
247   \@unusedoptionlist\@unusedoptionlist
248   \csname ds@\CurrentOption\endcsname}
249 \@onlypreamble\@use@option

```

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

```

250 (/2kernel)
251 (latexrelease)\IncludeInRelease{2017/01/01}%
252 (latexrelease)          {\@if@pti@ns}{Spaces in \ExecuteOptions}%
253 (*2kernel | latexrelease)
254 \def\ExecuteOptions#1{%

```

Use `\@fortmp` here as it is anyway cleared during `\@for` loop so does not change any existing names.

```

255   \edef\@fortmp{\zap@space#1 \@empty}%
256   \def\reserved@a##1\@nil{%
257     \@for\CurrentOption:=\@fortmp\do
258       {\csname ds@\CurrentOption\endcsname}%
259     \edef\CurrentOption{##1}}%
260   \expandafter\reserved@a\CurrentOption\@nil}
261 (/2kernel | latexrelease)
262 (latexrelease)\EndIncludeInRelease
263 (latexrelease)\IncludeInRelease{0000/00/00}%
264 (latexrelease)          {\@if@pti@ns}{Spaces in \ExecuteOptions}%
265 (latexrelease)\def\ExecuteOptions#1{%
266   (latexrelease)   \def\reserved@a##1\@nil{%
267     (latexrelease)   \@for\CurrentOption:=#1\do
268       (latexrelease)   {\csname ds@\CurrentOption\endcsname}%
269     (latexrelease)   \edef\CurrentOption{##1}}%
270   (latexrelease)   \expandafter\reserved@a\CurrentOption\@nil}
271 (latexrelease)\EndIncludeInRelease
272 (*2kernel)

```

273 `\@onlypreamble\ExecuteOptions`

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

`\documentclass` The main new-style class declaration.

```

274 \def\documentclass{%
275   \let\documentclass\@twoclasseserror
276   \ifcompatibility\else\let\usepackage\RequirePackage\fi
277   \@fileswithoptions\@clsextension}
278 \@onlypreamble\documentclass

```

`\documentstyle` 2.09 style class ‘style’ declaration.

```

279 \def\documentstyle{%
280   \makeatletter\input{latex209.def}\makeatother
281   \documentclass}
282 \@onlypreamble\documentstyle

```

`\RequirePackage` Load package if not already loaded.

```

283 \def\RequirePackage{%
284   \@fileswithoptions\@pkgextension}
285 \@onlypreamble\RequirePackage

\LoadClass Load class.
286 \def\LoadClass{%
287   \ifx\@current\@pkgextension
288     \@latex@error
289     {\noexpand\LoadClass in package file}%
290     {You may only use \noexpand\LoadClass in a class file.}%
291   \fi
292   \@fileswithoptions\@clsextension}
293 \@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.
294 \def\@loadwithoptions#1#2#3{%
295   \expandafter\let\csname opt@#3.#1\expandafter\endcsname
296     \csname opt@\@currname.\@current\endcsname
297   #2{#3}}
298 \@onlypreamble\@loadwithoptions

\LoadClassWithOptions Load class '#1' with the current option list.
299 \def\LoadClassWithOptions{%
300   \@loadwithoptions\@clsextension\LoadClass}
301 \@onlypreamble\LoadClassWithOptions

\RequirePackageWithOptions Load package '#1' with the current option list.
302 \def\RequirePackageWithOptions{%
303   \AtEndOfPackage{\let\@unprocessedoptions\relax}%
304   \@loadwithoptions\@pkgextension\RequirePackage}
305 \@onlypreamble\RequirePackageWithOptions

\usepackage To begin with, \usepackage produces an error. This is reset by \documentclass.
306 \def\usepackage#1#{%
307   \@latex@error
308   {\noexpand \usepackage before \string\documentclass}%
309   {\noexpand \usepackage may only appear in the document
310     preamble, i.e.,\MessageBreak
311     between \noexpand\documentclass and
312     \string\begin{document}.}%
313   \@gobble}
314 \@onlypreamble\usepackage

\NeedsTeXFormat Check that the document is running on the correct system.
315 \def\NeedsTeXFormat#1{%
316   \def\reserved@a{#1}%
317   \ifx\reserved@a\fmtname
318     \expandafter\@needsformat
319   \else
320     \@latex@error{This file needs format ``\reserved@a'%
321       \MessageBreak but this is ``\fmtname'}{%
322       The current input file will not be processed

```

```

323 further,\MessageBreak
324 because it was written for some other flavor of
325 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.

```

326 \endinput \fi}
327 \@onlypreamble\NeedsTeXFormat

328 \def\@needsformat{%
329 \ifnextchar[%
330 \@needsf@rmat
331 {}
332 \@onlypreamble\@needsformat

333 \def\@needsf@rmat[#1]{%
334 \ifl@t@r\fmtversion{#1}{}%
335 {\@latex@warning@no@line
336 {You have requested release `#1' of LaTeX,\MessageBreak
337 but only release `\'fmtversion' is available}}}
338 \@onlypreamble\@needsf@rmat

```

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

```

339 \def\zap@space#1 #2{%
340 #1%
341 \ifx#2\@empty\else\expandafter\zap@space\fi
342 #2}

```

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

```

343 \def\@fileswithoptions#1{%
344 \ifnextchar[%
345 {\@fileswith@ptions#1}%
346 {\@fileswith@ptions#1[]}}
347 \@onlypreamble\@fileswithoptions

348 \def\@fileswith@ptions#1[#2]#3{%
349 \ifnextchar[%
350 {\@fileswith@ptions#1[#{#2}]#3}%
351 {\@fileswith@ptions#1[#{#2}]#3[]}}
352 \@onlypreamble\@fileswith@ptions

```

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.

```

353 /2ekernel)
354 (latexrelease)\IncludeInRelease{2017/01/01}%
355 (latexrelease) {\@fileswith@ptions}{ifx tests in \@fileswith@ptions}%

```

```

356 <*2ekernel | latexrelease>
357 \def\@fileswith@pti@ns#1[#2]#3[#4]{%
358   \ifx#1\@clsextension
359     \ifx\@classoptionslist\relax
360       \xdef\@classoptionslist{\zap@space#2 \@empty}%
361       \def\reserved@a{%
362         \@onefilewithoptions#3[{#2}][#{4}]#1%
363         \@documentclasshook}%
364     \else
365       \def\reserved@a{%
366         \@onefilewithoptions#3[{#2}][#{4}]#1%
367       \fi
368     \else

```

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

```

369   \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)

```

370     \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.)

```

371       \ifx\@nnil##1\@nnil\else
372         \noexpand\@onefilewithoptions##1[{#2}][#{4}]%
373         \noexpand\@pkgextension
374       \fi
375       \expandafter\reserved@b
376     \fi}%
377     \edef\reserved@a{\zap@space#3 \@empty}%
378     \edef\reserved@a{\expandafter\reserved@b\reserved@a,\@nnil,}%
379   \fi
380   \reserved@a}
381 </2ekernel | latexrelease>

```

```

382 <latexrelease>\EndIncludeInRelease
383 <latexrelease>\IncludeInRelease{0000/00/00}%
384 <latexrelease>      {\@fileswith@pti@ns}{\ifx tests in \@fileswith@pti@ns}%
385 <latexrelease>\def\@fileswith@pti@ns#1[#2]#3[#4]{%
386 <latexrelease>  \ifx#1\@clsextension
387 <latexrelease>    \ifx\@classoptionslist\relax
388 <latexrelease>      \xdef\@classoptionslist{\zap@space#2 \@empty}%
389 <latexrelease>      \def\reserved@a{%
390 <latexrelease>        \@onefilewithoptions#3[{#2}][#{4}]#1%
391 <latexrelease>        \@documentclasshook}%
392 <latexrelease>    \else
393 <latexrelease>      \def\reserved@a{%
394 <latexrelease>        \@onefilewithoptions#3[{#2}][#{4}]#1}%
395 <latexrelease>      \fi
396 <latexrelease>    \else
397 <latexrelease>      \def\reserved@b##1,{%
398 <latexrelease>        \ifx\@nil##1\relax\else
399 <latexrelease>          \ifx\relax##1\relax\else

```

```

400 \latexrelease> \noexpand\@onefilewithoptions##1[#{#2}][#{#4}]%
401 \latexrelease> \noexpand\@pkgextension
402 \latexrelease> \fi
403 \latexrelease> \expandafter\reserved@b
404 \latexrelease> \fi}%
405 \latexrelease> \edef\reserved@a{\zap@space#3 \@empty}%
406 \latexrelease> \edef\reserved@a{%
407 \latexrelease> \expandafter\reserved@b\reserved@a,\@nil,}%
408 \latexrelease> \fi
409 \latexrelease> \reserved@a}
410 \latexrelease>\EndIncludeInRelease
411 (*2ekernel)
412 \@onlypreamble\@fileswith@ptions

```

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

```

413 \def\@onefilewithoptions#1[#2][#3]#4{%
414   \@pushfilename
415   \xdef\@currname{#1}%
416   \global\let\@currentx#4%
417   \expandafter\let\csname\@currname.\@currentx-h@@k\endcsname\@empty
418   \let\CurrentOption\@empty
419   \@reset@ptions
420   \makeatletter

```

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

```

421 \def\reserved@a{%
422   \@ifl@aded\@currentx{#1}%
423   {\@if@ptions\@currentx{#1}{#2}{}}%
424   {\@latex@error
425     {Option clash for \@cls@pkg\space #1}%
426     {The package #1 has already been loaded
427       with options:\MessageBreak
428       \space\space\@optionlist{#1.\@currentx}}\MessageBreak
429     There has now been an attempt to load it
430     with options:\MessageBreak
431     \space\space[#2]\MessageBreak
432     Adding the global options:\MessageBreak
433     \space\space
434     \@optionlist{#1.\@currentx},#2\MessageBreak
435     to your \noexpand\documentclass declaration may fix this.%
436     \MessageBreak
437     Try typing \space <return> \space to proceed.}}}%
438   {\@pass@ptions\@currentx{#2}{#1}%
439   \global\expandafter
440   \let\csname ver@\@currname.\@currentx\endcsname\@empty
441   \InputIfFileExists
442     {\@currname.\@currentx}%
443     {}%
444     {\@missingfileerror\@currname\@currentx}%

```

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

```

445   \let\@unprocessedoptions\@unprocessedoptions

```

```

446 \csname\@currname.\@currentx-h@@k\endcsname
447 \expandafter\let\csname\@currname.\@currentx-h@@k\endcsname
448 \undefined
449 \@unprocessedoptions}%
450 \ifl@ter\@currentx{#1}{#3}{}%
451 {\@latex@warning@no@line
452 {You have requested,\on@line,
453 version\MessageBreak
454 `#3' of \@cls@pkg\space #1,\MessageBreak
455 but only version\MessageBreak
456 \csname ver@#1.\@currentx\endcsname'\MessageBreak
457 is available}}%
458 \ifx\@currentx\@clsextension\let\LoadClass\@twoloadclasserror\fi
459 \@popfilename
460 \@reset@ptions}%
461 \reserved@a}
462 \@onlypreamble\@onefilewithoptions

@@fileswith@ptions Save the definition (for error checking).
463 \let@@fileswith@ptions\@fileswith@ptions
464 \@onlypreamble\@@fileswith@ptions

\@reset@ptions Reset the default option, and clear lists of declared options.
465 \def\@reset@ptions{%
466 \global\ifx\@currentx\@clsextension
467 \let\default@ds\OptionNotUsed
468 \else
469 \let\default@ds\unknownoptionerror
470 \fi
471 \global\let\ds@\@empty
472 \global\let\@declaredoptions\@empty}
473 \@onlypreamble\@reset@ptions

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
474 \ifx\@begindocumenthook\undefined
475 \let\@begindocumenthook\@empty
476 \fi
477 \let\@enddocumenthook\@empty

\g@addto@macro Globally add to the end of a macro.
478 \long\def\g@addto@macro#1#2{%
479 \begingroup
480 \toks@\expandafter{#1#2}%
481 \xdef#1{\the\toks@}%
482 \endgroup}

```

```

\AtEndOfPackage The access functions.
\AtEndOfClass 483 \def\AtEndOfPackage{%
\AtBeginDocument 484 \expandafter\g@addto@macro\csname\@currname.\@currentx-h@@k\endcsname}
\AtEndDocument 485 \let\AtEndOfClass\AtEndOfPackage
486 \@onlypreamble\AtEndOfPackage
487 \@onlypreamble\AtEndOfClass

488 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
489 \def\AtEndDocument{\g@addto@macro\@enddocumenthook}
490 \@onlypreamble\AtBeginDocument

\@cls@pkg The current file type.
491 \def\@cls@pkg{%
492 \ifx\@currentx\@clsextension
493 document class%
494 \else
495 package%
496 \fi}
497 \@onlypreamble\@cls@pkg

\@unknownoptionerror Bad option.
498 \def\@unknownoptionerror{%
499 \@latex@error
500 {Unknown option '\CurrentOption' for \@cls@pkg\space'\@currname'}%
501 {The option '\CurrentOption' was not declared in
502 \@cls@pkg\space'\@currname', perhaps you\MessageBreak
503 misspelled its name.
504 Try typing \space <return>
505 \space to proceed.}}
506 \@onlypreamble\@unknownoptionerror

\@unprocessedoptions Declare an error for each option, unless a \ProcessOptions occurred.
507 \def\@unprocessedoptions{%
508 \ifx\@currentx\@pkgextension
509 \edef\@curroptions{\@optionlist{\@currname.\@currentx}}%
510 \@for\CurrentOption:=\@curroptions\do{%
511 \ifx\CurrentOption\@empty\else\@unknownoptionerror\fi}%
512 \fi}
513 \@onlypreamble\@unprocessedoptions
514 \@onlypreamble\@unprocessedoptions

\@badrequireerror \RequirePackage or \LoadClass occurs in the options section.
515 \def\@badrequireerror#1[#2]#3[#4]{%
516 \@latex@error
517 {\noexpand\RequirePackage or \noexpand\LoadClass
518 in Options Section}%
519 {The \@cls@pkg\space'\@currname' is defective.\MessageBreak
520 It attempts to load '#3' in the options section, i.e.,\MessageBreak
521 between \noexpand\DeclareOption and \string\ProcessOptions.}}
522 \@onlypreamble\@badrequireerror

\@twoloadclasserror Two \LoadClass in a class.
523 \def\@twoloadclasserror{%
524 \@latex@error

```

```

525     {Two \noexpand\LoadClass commands}%
526     {You may only use one \noexpand\LoadClass in a class file}}
527 \@onlypreamble\@twoloadclasserror

\@twoclasseserror Two \documentclass or \documentstyle.
528 \def\@twoclasseserror#1#{%
529   \@latex@error
530   {Two \noexpand\documentclass or \noexpand\documentstyle commands}%
531   {The document may only declare one class.}\@gobble}
532 \@onlypreamble\@twoclasseserror

```

4.2 Providing shipment

```

\two@digits Prefix a number less than 10 with '0'.
533 \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.

534 \begingroup%
535 \@tempcnta=1
536 \loop
537   \catcode\@tempcnta=12 %
538   \advance\@tempcnta\@ne %
539   \ifnum\@tempcnta<32 %
540     \repeat %
541     \catcode\*=11 %
542     \catcode\^^M\active%
543     \catcode\^^L\active\let^^L\relax%
544     \catcode\^^I\active%

545 \gdef\filecontents{\@tempswatrue\filecontents}%
546 \gdef\filecontents*{\@tempswafalse\filecontents}%

547 \gdef\filecontents#1{%
548   \openin\@inputcheck#1 %
549   \ifeof\@inputcheck%
550     \@latex@warning@no@line%
551     {Writing file \@currdir#1'}%

552   \chardef\reserved@c15 %
553   \ch@ck7\reserved@c\write%
554   \immediate\openout\reserved@c#1\relax%
555   \else%

556   \closein\@inputcheck%
557   \@latex@warning@no@line%
558     {File `#1' already exists on the system.\MessageBreak%
559     Not generating it from this source}%

560   \let\write@gobbletwo%
561   \let\closeout@gobble%
562   \fi%
563   \if@tempswa%

564   \immediate\write\reserved@c{%
565     \@percentchar\@percentchar\space%
566     \expandafter\@gobble\string\LaTeX2e file `#1'^J%

```

```

567      \@percentchar\@percentchar\space generated by the %
568      ``\@currentvir' \expandafter\@gobblefour\string\newenvironment^^J%
569      \@percentchar\@percentchar\space from source ``\jobname' on %
570      \number\year/\two@digits\month/\two@digits\day.^^J%
571      \@percentchar\@percentchar}%
572 \fi%
573 \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.

```

574 \count@ 128\relax%
575 \loop%
576   \catcode\count@ 11\relax%
577   \advance\count@ \@ne%
578   \ifnum\count@<\@ccclvi%
579 \repeat%
580 \edef\E{\@backslashchar end\string{\@currentvir\string}}%
581 \edef\reserved@b{%
582   \def\noexpand\reserved@b%
583     #####1\E####2\E####3\relax}%
584 \reserved@b{%
585   \ifx\relax###3\relax%

```

There was no `\end{filecontents}`

```

586   \immediate\write\reserved@c{##1}%
587 \else%

```

There was a `\end{filecontents}`, so stop this time.

```

588   \edef^^M{\noexpand\end{\@currentvir}}%
589   \ifx\relax##1\relax%
590 \else%

```

Text before the `\end`, write it with a warning.

```

591   \@latex@warning{Writing text `##1' before %
592     \string\end{\@currentvir}\MessageBreak as last line of #1}%
593   \immediate\write\reserved@c{##1}%
594 \fi%
595   \ifx\relax##2\relax%
596 \else%

```

Text after the `\end`, ignore it with a warning.

```

597   \@latex@warning{%
598     Ignoring text `##2' after \string\end{\@currentvir}}%
599 \fi%
600 \fi%
601 ^^M}%
602 \catcode``^^L\active%
603 \let\L\@undefined%
604 \def^^L{\@ifundefined L^^J^^J^^J}%
605 \catcode``^^I\active%
606 \let\I\@undefined%
607 \def^^I{\@ifundefined I\space\space}%
608 \catcode``^^M\active%
609 \edef^^M##1^^M{%
610   \noexpand\reserved@b##1\E\E\relax}}%
611 \endgroup%

```

```

612 \begingroup
613 \catcode`\|=\catcode`\%
614 \catcode`\%=12
615 \catcode`\*=11
616 \gdef\@percentchar{%}
617 \gdef\endfilecontents{|
618   \immediate\closeout\reserved@c
619   \def\T##1##2##3{|
620     \ifx##1\@undefined\else
621       \@latex@warning@no@line{##2 has been converted to Blank ##3e}|
622     \fi}|
623   \T\L{Form Feed}{Lin}|
624   \T\I{Tab}{Spac}|
625   \immediate\write\@unused{}}
626 \global\let\endfilecontents*\endfilecontents
627 \@onlypreamble\filecontents
628 \@onlypreamble\endfilecontents
629 \@onlypreamble\filecontents*
630 \@onlypreamble\endfilecontents*
631 \endgroup
632 \@onlypreamble\filecontents

```

5 Package/class rollback mechanism

```

633 </2kernel>
634 <*2kernel | 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.

```

635 <*tracerollback>
636 %\let\pkgcls@debug\typeout
637 \let\pkgcls@debug\@gobble
638 </tracerollback>

```

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

```

639 \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 `TEX` largest legal number. While loading a file this can be used to provide conditionals that select code based on the request.

```

640 \ifx\pkgcls@targetdate\@undefined

```

```

641 \newcount\pkgcls@targetdate
642 \fi
643 \let\pkgcls@targetlabel\@empty
644 \def\pkgcls@innerdate{\maxdimen}

\pkgcls@candidate When looping through the \DeclareRelease declarations we record if the release
\pkgcls@releasedate 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.
645 \let\pkgcls@candidate\@empty
646 \let\pkgcls@releasedate\@empty

\load@onefilewithoptions the best place to add the rollback code is at the point where \@onefilewithoptions
\@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.
647 \ifx\load@onefilewithoptions\@undefined
648 \let\load@onefilewithoptions\@onefilewithoptions
649 \def\@onefilewithoptions#1[#2][#3]#4{%

First a bit of tracing normally disabled.
650 (*tracrollback)
651 \pkgcls@debug{--- File loaded request (\noexpand\usepackage or ...)}%
652 \pkgcls@debug{\@spaces 1: #1}%
653 \pkgcls@debug{\@spaces 2: #2}%
654 \pkgcls@debug{\@spaces 3: #3}%
655 \pkgcls@debug{\@spaces 4: #4}%
656 /tracrollback)

Two of the arguments are needed later on in error/warning messages so we save
them.
657 \def\pkgcls@name{#1}% % for info message
658 \def\pkgcls@arg {#3}% % for info message

then we parse the final optional argument to determine if there is a spe-
cific rollback request for the current file. This will set \pkgcls@targetdate,
\pkgcls@targetlabel and \pkgcls@mindate.
659 \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:
660 \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 \pkgcls@mindate.
661 \load@onefilewithoptions#1[#2][\pkgcls@mindate]#4%
662 }
663 \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 the 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>	→	1	<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.

```
664 \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.

```
665   \ifx\@nil#1\@nil
666     \pkgcls@targetdate\requestedLaTeXdate\relax
667     \let\pkgcls@targetlabel\@empty
668     \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.

```
669   \else
670     \pkgcls@parse@date@arg#1=\@nil\relax
671   \fi
672 }
```

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

```
673 \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.

```
674   \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.

```
675 \ifx\@nil#1\@nil
676 \@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 =).

```
677 \ifnum \pkgcls@targetdate=\z@
678 \pkgcls@targetdate\@ne
679 \def\pkgcls@innerdate{\maxdimen}%
680 \pkgcls@parse@date@arg@version#2%
681 \else
682 \edef\pkgcls@innerdate{\the\pkgcls@targetdate}%
683 \fi
684 \let\pkgcls@mindate\@empty
685 \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`):

```
686 \requestedLaTeXdate\relax
687 \let\pkgcls@targetlabel\@empty
688 \def\pkgcls@innerdate{\maxdimen}%
689 \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 an error.

```
690 \ifnum \pkgcls@targetdate > \z@
691 \ifnum \@parse@version0#1//00\@nil > \pkgcls@targetdate
692 \@latex@error{Suspicious rollback/min-date date given}%
693 {There is a minimal date of #1 specified for
694 \cls@pkg\space'\pkgcls@name'.\MessageBreak
695 But this is in conflict
696 with a rollback request to \requestedpatchdate,
697 so something\MessageBreak
698 is wrong here. Continue and I
699 ignore the minimal date request.}%
700 \fi
701 \fi
702 \fi
703 }
```

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

```
704 \def\pkgcls@parse@date@arg@version#1={%
705 \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 $\langle pkg/cls-name \rangle - \langle date \rangle . \langle extension \rangle$ but this is not enforced and through this argument one can overwrite it.

```

706 \def\DeclareRelease#1#2#3{%
707   \ifnum\pkgcls@targetdate>\z@ % some sort of rollback request
708   (*tracereollback)
709     \pkgcls@debug{---\string\DeclareRelease:}%
710     \pkgcls@debug{\@spaces 1: #1}%
711     \pkgcls@debug{\@spaces 2: #2}%
712     \pkgcls@debug{\@spaces 3: #3}%
713   /tracereollback)

```

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.

```

714   \ifx\@nil#2\@nil
715     \ifnum\pkgcls@targetdate=\@ne % named request
716       \def\reserved@a{#1}%
717       \ifx\pkgcls@targetlabel\reserved@a
718         \pkgcls@use@this@release{#3}{}%
719       (*tracereollback)
720     \else
721       \pkgcls@debug{Label doesn't match}%
722     /tracereollback)
723   \fi
724   (*tracereollback)
725   \else
726     \pkgcls@debug{Date request: ignored}%
727   /tracereollback)
728   \fi
729   \else

```

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

```

730   \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.

```

731   \ifnum\@parse@version#2//00\@nil
732     >\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.

```

733   \ifx\pkgcls@candidate\@empty
734     \pkgcls@rollbackdate@error{#2}%
735     \pkgcls@use@this@release{#3}{#2}%
736   \else
737     \pkgcls@use@this@release\pkgcls@candidate
738     \pkgcls@releasedate
739   \fi
740   \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.

```

741         \def\pkgcls@candidate{#3}%
742         \def\pkgcls@releasedate{#2}%
743 (*tracerollback)
744         \pkgcls@debug{New candidate: #3}%
745 
```

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.

```

748         \def\reserved@a{#1}%
749         \ifx\pkgcls@targetlabel\reserved@a
750         \pkgcls@use@this@release{#3}{#2}%
751 (*tracerollback)
752     \else
753         \pkgcls@debug{Label doesn't match}%
754 
```

`\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.

```

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

```

Before that we record the selection made inside the transcript.

```

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

```

762     \pkgcls@targetdate\z@
763     \@@input #1\relax
764     \endinput
765 }

```

`\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

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.

debugging. The second argument is the date that corresponds to this file and it is used as part of the message.

```

766 \def\pkgcls@show@selection#1#2{%
767 (*traceroollback)
768 \pkgcls@debug{Result: use #1}%
769 /traceroollback)
770 \GenericInfo
771 {\@spaces\@spaces\space}{Rollback for
772 \@cls@pkg\space'\@currname' requested ->
773 \ifnum\pkgcls@targetdate>\@ne
774 date
775 \ifnum\requestedLaTeXdate=\pkgcls@targetdate
776 \requestedpatchdate
777 \else
778 \expandafter\@gobble\pkgcls@arg
779 \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.

```

780 Best approximation is
781 \else
782 version '\pkgcls@targetlabel'.\MessageBreak
783 This corresponds to
784 \fi
785 \ifx\@nil#2\@nil
786 a special release%
787 \else
788 the release introduced on #2%
789 \fi
790 \@gobble}%
791 }

```

\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.

```

792 \def\pkgcls@rollbackdate@error#1{%
793 \@latex@error{Suspicious rollback date given}%
794 {The \@cls@pkg\space'\@currname' claims that it
795 came into existence on #1 which\MessageBreak
796 is after your requested rollback date --- so
797 something is wrong here.\MessageBreak
798 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.

```

799 \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).

```

800 \ifnum\pkgcls@targetdate>\z@ % some sort of rollback request
801 (*traceroollback)

```

```

802 \pkgcls@debug{---DeclareCurrentRelease}%
803 \pkgcls@debug{ 1: #1}%
804 \pkgcls@debug{ 2: #2}%
805 \traceroollback)

```

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.

```

806 \ifnum\pkgcls@targetdate>\@ne % a date request
807 \ifnum\@parse@version#2//00\@nil
808 >\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).

```

809
810 \ifx\pkgcls@candidate\@empty
811 \pkgcls@rollbackdate@error{#2}%
812 \else
813 \pkgcls@use@this@release\pkgcls@candidate
814 \pkgcls@releasedate
815 \fi

```

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

```

816 \else
817 \pkgcls@show@selection{current version}{#2}%
818 \fi
819 \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.

```

820 \def\reserved@a{#1}%
821 \ifx\pkgcls@targetlabel\reserved@a
822 \pkgcls@show@selection{current version}{#2}%
823 \else
824 \@latex@error{Requested version '\pkgcls@targetlabel' for
825 \cls@pkg\space'\@currname' is unknown}\@ehc
826 \fi
827 \fi
828 \fi
829 }

```

\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.

```

830 \long\def\IfTargetDateBefore#1{%
831 \ifnum\pkgcls@innerdate <%
832 \expandafter\@parse@version\expandafter0#1//00\@nil

```

```

833     \typeout{Exclude code introduced on #1}%
834     \expandafter\@firstoftwo
835   \else
836     \typeout{Include code introduced on #1}%
837     \expandafter\@secondoftwo
838   \fi
839 }

840 </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}`.

```

841 <*afterpreamble>
842 \NeedsTeXFormat{LaTeX2e}
843 \ProvidesPackage{pkgindoc}
844     [1994/10/20 v1.1 Package Interface in Document (DPC)]
845 \def\reserved@a#1\do\@classoptionslist#2\do\filecontents#3\relax{%
846   \gdef\@preamblecmds{#1#3}}
847 \expandafter\reserved@a\@preamblecmds\relax
848 </afterpreamble>

```