The main structure of documents

Frank Mittelbach Chris Rowley Alan Jeffrey
David Carlisle

2018/03/27

This file is maintained by the LaTeX 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 LaTeX 2$_\varepsilon$ 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 LaTeX 2$_\varepsilon$ and LaTeX2.09 is that LaTeX 2$_\varepsilon$ 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 LaTeX2.09 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 LaTeX2.09 compatibility mode. As far as most packages are concerned, this only affects the warnings and errors LaTeX 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` To find out if a package has already been loaded, use `\@ifpackageloaded{`⟨*package*⟩`}{`⟨*true*⟩`}{`⟨*false*⟩`}`.

`\@ifpackagelater` `\@ifclasslater` 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*⟩`}`.

`\@ifpackagewith` `\@ifclasswith` 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` 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 LaTeX2.09 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 ⟨*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
```

`\@unusedoptionlist` List of options of the main class that haven't been declared or loaded as class option files.

```
\let\@unusedoptionlist\@empty
\@onlypreamble\@unusedoptionlist
```

`\CurrentOption` Name of current package or option.

```
\let\CurrentOption\@empty
```

`\@currname` Name of current package or option.

```
\let\@currname\@empty
```

`\@currext` The current file extension.

```
\global\let\@currext=\@empty
```

`\@clsextension` The two possible values of `\@currext`.
`\@pkgextension`

```
\def\@clsextension{cls}
\def\@pkgextension{sty}
\@onlypreamble\@clsextension
\@onlypreamble\@pkgextension
```

`\@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.

```
\def\@pushfilename{%
  \xdef\@currnamestack{%
    {\@currname}%
    {\@currext}%
    {\the\catcode`\@}%
    \@currnamestack}}
\@onlypreamble\@pushfilename
\def\@popfilename{\expandafter\@p@pfilename\@currnamestack\@nil}
\@onlypreamble\@popfilename
\def\@p@pfilename#1#2#3#4\@nil{%
  \gdef\@currname{#1}%
  \gdef\@currext{#2}%
  \catcode`\@#3\relax
  \gdef\@currnamestack{#4}}
\@onlypreamble\@p@pfilename
\gdef\@currnamestack{}
\@onlypreamble\@currnamestack
```

`\@ptionlist` Returns the option list of the file.

```
\def\@ptionlist#1{%
  \@ifundefined{opt@#1}\@empty{\csname opt@#1\endcsname}}
\@onlypreamble\@ptionlist
```

`\@ifpackageloaded` `\@ifpackageloaded{`⟨*name*⟩`}` Checks to see whether a file has been loaded.
`\@ifclassloaded`

```
\def\@ifpackageloaded{\@ifl@aded\@pkgextension}
\def\@ifclassloaded{\@ifl@aded\@clsextension}
\@onlypreamble\@ifpackageloaded
\@onlypreamble\@ifclassloaded
```

```
\def\@ifl@aded#1#2{%
  \expandafter\ifx\csname ver@#2.#1\endcsname\relax
    \expandafter\@secondoftwo
  \else
    \expandafter\@firstoftwo
  \fi}
\@onlypreamble\@ifl@aded
```

`\@ifpackagelater` `\@ifclasslater` `\@ifpackagelater{`⟨*name*⟩`}{YYYY/MM/DD}` Checks that the package loaded is more recent than the given date.

```
\def\@ifpackagelater{\@ifl@ter\@pkgextension}
\def\@ifclasslater{\@ifl@ter\@clsextension}
\@onlypreamble\@ifpackagelater
\@onlypreamble\@ifclasslater
```

```
\def\@ifl@ter#1#2{%
  \expandafter\@ifl@t@r
    \csname ver@#2.#1\endcsname}
\@onlypreamble\@ifl@ter
⟨/2ekernel⟩
```

This internal macro is also used in `\NeedsTeXFormat`.

```
⟨latexrelease⟩\IncludeInRelease{2018/04/01}%
⟨latexrelease⟩                  {\@ifl@t@r}{Guard against bad input}%
⟨*2ekernel | latexrelease⟩
\def\@ifl@t@r#1#2{%
  \ifnum\expandafter\@parse@version\expandafter0#1//00\@nil<%
        \expandafter\@parse@version\expandafter0#2//00\@nil
    \expandafter\@secondoftwo
  \else
    \expandafter\@firstoftwo
  \fi}
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩                  {\@ifl@t@r}{Guard against bad input}%
⟨latexrelease⟩\def\@ifl@t@r#1#2{%
⟨latexrelease⟩  \ifnum\expandafter\@parse@version#1//00\@nil<%
⟨latexrelease⟩        \expandafter\@parse@version#2//00\@nil
⟨latexrelease⟩    \expandafter\@secondoftwo
⟨latexrelease⟩  \else
⟨latexrelease⟩    \expandafter\@firstoftwo
⟨latexrelease⟩  \fi}
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

```
\@onlypreamble\@ifl@t@r
```

```
⟨/2ekernel⟩
⟨*2ekernel | latexreleasefirst⟩
\def\@parse@version#1/#2/#3#4#5\@nil{%
\@parse@version@dash#1-#2-#3#4\@nil
}
```

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

```
\def\@parse@version@dash#1-#2-#3#4#5\@nil{%
  \if\relax#2\relax\else#1\fi#2#3#4 }
⟨/2ekernel | latexreleasefirst⟩
⟨*2ekernel⟩
```

\@ifpackagewith
\@ifclasswith

\@ifpackagewith{⟨*name*⟩}{⟨*option-list*⟩} Checks that ⟨*option-list*⟩ is a subset of the options **with** which ⟨*name*⟩ was loaded.

```
\def\@ifpackagewith{\@if@ptions\@pkgextension}
\def\@ifclasswith{\@if@ptions\@clsextension}
\@onlypreamble\@ifpackagewith
\@onlypreamble\@ifclasswith

\def\@if@ptions#1#2{%
  \@expandtwoargs\@if@pti@ns{\@ptionlist{#2.#1}}}
\@onlypreamble\@if@ptions
```

Probably shouldn't use \CurrentOption here... (changed to \reserved@b.)

```
⟨/2ekernel⟩
⟨latexrelease⟩\IncludeInRelease{2017/01/01}%
⟨latexrelease⟩                  {\@if@pti@ns}{Spaces in option clash check}%
⟨*2ekernel | latexrelease⟩
\def\@if@pti@ns#1#2{%
 \let\reserved@a\@firstoftwo

 \edef\reserved@b{\zap@space#2 \@empty}%
 \@for\reserved@b:=\reserved@b\do{%
   \ifx\reserved@b\@empty
   \else
     \expandafter\in@\expandafter{\expandafter,\reserved@b,}{,#1,}%
     \ifin@
     \else
       \let\reserved@a\@secondoftwo
     \fi
   \fi
 }%
 \reserved@a}
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩                  {\@if@pti@ns}{Spaces in option clash check}%
⟨latexrelease⟩\def\@if@pti@ns#1#2{%
⟨latexrelease⟩ \let\reserved@a\@firstoftwo
⟨latexrelease⟩ \@for\reserved@b:=#2\do{%
⟨latexrelease⟩  \ifx\reserved@b\@empty
⟨latexrelease⟩   \else
⟨latexrelease⟩   \expandafter\in@\expandafter
⟨latexrelease⟩                   {\expandafter,\reserved@b,}{,#1,}%
⟨latexrelease⟩    \ifin@
⟨latexrelease⟩    \else
⟨latexrelease⟩     \let\reserved@a\@secondoftwo
⟨latexrelease⟩    \fi
⟨latexrelease⟩  \fi
⟨latexrelease⟩ }%
⟨latexrelease⟩ \reserved@a}
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

```
\@onlypreamble\@if@pti@ns
```

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

```
\def\ProvidesPackage#1{%
  \xdef\@gtempa{#1}%
  \ifx\@gtempa\@currname\else
    \@latex@warning@no@line{You have requested
      \@cls@pkg\space`\@currname',\MessageBreak
       but the \@cls@pkg\space provides `#1'}%
  \fi
  \@ifnextchar[\@pr@videpackage{\@pr@videpackage[]}%]
\@onlypreamble\ProvidesPackage
\def\@pr@videpackage[#1]{%
  \expandafter\xdef\csname ver@\@currname.\@currext\endcsname{#1}%
  \ifx\@currext\@clsextension
    \typeout{Document Class: \@gtempa\space#1}%
  \else
    \wlog{Package: \@gtempa\space#1}%
  \fi}
\@onlypreamble\@pr@videpackage
```

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

```
\let\ProvidesClass\ProvidesPackage
\@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`

```
\def\ProvidesFile#1{%
  \begingroup
    \catcode`\ 10 %
    \ifnum \endlinechar<256 %
      \ifnum \endlinechar>\m@ne
        \catcode\endlinechar 10 %
      \fi
    \fi
    \@makeother\/%
    \@makeother\&%
    \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` `\PassOptionsToClass` If the package has been loaded, we check that it was first loaded with the options. Otherwise we add the option list to that of the package.

```
\def\@pass@ptions#1#2#3{%
  \expandafter\xdef\csname opt@#3.#1\endcsname{%
    \@ifundefined{opt@#3.#1}\@empty
```

```
      {\csname opt@#3.#1\endcsname,}%
    \zap@space#2 \@empty}}
\@onlypreamble\@pass@ptions
\def\PassOptionsToPackage{\@pass@ptions\@pkgextension}
\def\PassOptionsToClass{\@pass@ptions\@clsextension}
\@onlypreamble\PassOptionsToPackage
\@onlypreamble\PassOptionsToClass
```

`\DeclareOption` `\DeclareOption*`

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

```
\def\DeclareOption{%
  \let\@fileswith@pti@ns\@badrequireerror
  \@ifstar\@defdefault@ds\@declareoption}
\long\def\@declareoption#1#2{%
   \xdef\@declaredoptions{\@declaredoptions,#1}%
   \toks@{#2}%
   \expandafter\edef\csname ds@#1\endcsname{\the\toks@}}
\long\def\@defdefault@ds#1{%
  \toks@{#1}%
  \edef\default@ds{\the\toks@}}
\@onlypreamble\DeclareOption
\@onlypreamble\@declareoption
\@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.

```
\def\OptionNotUsed{%
  \ifx\@currext\@clsextension
    \xdef\@unusedoptionlist{%
      \ifx\@unusedoptionlist\@empty\else\@unusedoptionlist,\fi
      \CurrentOption}%
  \fi}
\@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*`.

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

`\ProcessOptions` `\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.

```
\def\ProcessOptions{%
  \let\ds@\@empty
  \edef\@curroptions{\@ptionlist{\@currname.\@currext}}%
  \@ifstar\@xprocess@ptions\@process@ptions}
\@onlypreamble\ProcessOptions
```

```
\def\@process@ptions{%
  \@for\CurrentOption:=\@declaredoptions\do{%
    \ifx\CurrentOption\@empty\else
      \@expandtwoargs\in@{,\CurrentOption,}{%
         ,\ifx\@currext\@clsextension\else\@classoptionslist,\fi
         \@curroptions,}%
      \ifin@
        \@use@ption
        \expandafter\let\csname ds@\CurrentOption\endcsname\@empty
      \fi
    \fi}%
  \@process@pti@ns}
\@onlypreamble\@process@ptions
\def\@xprocess@ptions{%
  \ifx\@currext\@clsextension\else
    \@for\CurrentOption:=\@classoptionslist\do{%
      \ifx\CurrentOption\@empty\else
        \@expandtwoargs\in@{,\CurrentOption,}{,\@declaredoptions,}%
        \ifin@
          \@use@ption
          \expandafter\let\csname ds@\CurrentOption\endcsname\@empty
        \fi
      \fi}%
  \fi
  \@process@pti@ns}
\@onlypreamble\@xprocess@ptions
```

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

```
\def\@process@pti@ns{%
  \@for\CurrentOption:=\@curroptions\do{%
    \@ifundefined{ds@\CurrentOption}%
      {\@use@ption
       \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`.

```
      \@use@ption}%
```

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

```
  \@for\CurrentOption:=\@declaredoptions\do{%
    \expandafter\let\csname ds@\CurrentOption\endcsname\relax}%
  \let\CurrentOption\@empty
  \let\@fileswith@pti@ns\@@fileswith@pti@ns
  \AtEndOfPackage{\let\@unprocessedoptions\relax}}
\@onlypreamble\@process@pti@ns
```

`\@options` `\@options` is a synonym for `\ProcessOptions*` for upward compatibility with LaTeX2.09 style files.

```
\def\@options{\ProcessOptions*}
\@onlypreamble\@options
```

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

```
\def\@use@ption{%
  \@expandtwoargs\@removeelement\CurrentOption
  \@unusedoptionlist\@unusedoptionlist
  \csname ds@\CurrentOption\endcsname}
\@onlypreamble\@use@ption
```

`\ExecuteOptions` `\ExecuteOptions{`⟨*option-list*⟩`}` executes the code declared for each option.

```
⟨/2ekernel⟩
⟨latexrelease⟩\IncludeInRelease{2017/01/01}%
⟨latexrelease⟩                  {\@if@pti@ns}{Spaces in \ExecuteOptions}%
⟨*2ekernel | latexrelease⟩
\def\ExecuteOptions#1{%
```

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

```
  \edef\@fortmp{\zap@space#1 \@empty}%
  \def\reserved@a##1\@nil{%
    \@for\CurrentOption:=\@fortmp\do
              {\csname ds@\CurrentOption\endcsname}%
    \edef\CurrentOption{##1}}%
  \expandafter\reserved@a\CurrentOption\@nil}
⟨/2ekernel | latexrelease⟩
⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩                  {\@if@pti@ns}{Spaces in \ExecuteOptions}%
⟨latexrelease⟩\def\ExecuteOptions#1{%
⟨latexrelease⟩  \def\reserved@a##1\@nil{%
⟨latexrelease⟩    \@for\CurrentOption:=#1\do
⟨latexrelease⟩              {\csname ds@\CurrentOption\endcsname}%
⟨latexrelease⟩    \edef\CurrentOption{##1}}%
⟨latexrelease⟩  \expandafter\reserved@a\CurrentOption\@nil}
⟨latexrelease⟩\EndIncludeInRelease
⟨*2ekernel⟩
```

```
\@onlypreamble\ExecuteOptions
```

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

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

```
\def\documentclass{%
  \let\documentclass\@twoclasseserror
  \if@compatibility\else\let\usepackage\RequirePackage\fi
  \@fileswithoptions\@clsextension}
\@onlypreamble\documentclass
```

`\documentstyle` 2.09 style class 'style' declaration.

```
\def\documentstyle{%
  \makeatletter\input{latex209.def}\makeatother
  \documentclass}
\@onlypreamble\documentstyle
```

`\RequirePackage` Load package if not already loaded.

```
\def\RequirePackage{%
  \@fileswithoptions\@pkgextension}
\@onlypreamble\RequirePackage
```

`\LoadClass` Load class.

```
\def\LoadClass{%
  \ifx\@currext\@pkgextension
     \@latex@error
      {\noexpand\LoadClass in package file}%
      {You may only use \noexpand\LoadClass in a class file.}%
  \fi
  \@fileswithoptions\@clsextension}
\@onlypreamble\LoadClass
```

`\@loadwithoptions` Pass the current option list on to a class or package. `#1` is `\@`*cls-or-pkg*`extension`, `#2` is `\RequirePackage` or `\LoadClass`, `#3` is the class or package to be loaded.

```
\def\@loadwithoptions#1#2#3{%
  \expandafter\let\csname opt@#3.#1\expandafter\endcsname
       \csname opt@\@currname.\@currext\endcsname
   #2{#3}}
\@onlypreamble\@loadwithoptions
```

`\LoadClassWithOptions` Load class '`#1`' with the current option list.

```
\def\LoadClassWithOptions{%
  \@loadwithoptions\@clsextension\LoadClass}
\@onlypreamble\LoadClassWithOptions
```

`\RequirePackageWithOptions` Load package '`#1`' with the current option list.

```
\def\RequirePackageWithOptions{%
  \AtEndOfPackage{\let\@unprocessedoptions\relax}%
  \@loadwithoptions\@pkgextension\RequirePackage}
\@onlypreamble\RequirePackageWithOptions
```

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

```
\def\usepackage#1#{%
  \@latex@error
    {\noexpand \usepackage before \string\documentclass}%
    {\noexpand \usepackage may only appear in the document
      preamble, i.e.,\MessageBreak
      between \noexpand\documentclass and
      \string\begin{document}.}%
  \@gobble}
\@onlypreamble\usepackage
```

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

```
\def\NeedsTeXFormat#1{%
  \def\reserved@a{#1}%
  \ifx\reserved@a\fmtname
    \expandafter\@needsformat
  \else
     \@latex@error{This file needs format `\reserved@a'%
       \MessageBreak but this is `\fmtname'}{%
       The current input file will not be processed
```

```
        further,\MessageBreak
        because it was written for some other flavor of
        TeX.\MessageBreak\@ehd}%
```

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

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

\def\@needsformat{%
  \@ifnextchar[%]
    \@needsf@rmat
    {}}
\@onlypreamble\@needsformat

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

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

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

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

```
\def\@fileswithoptions#1{%
  \@ifnextchar[%]
    {\@fileswith@ptions#1}%
    {\@fileswith@ptions#1[]}}
\@onlypreamble\@fileswithoptions

\def\@fileswith@ptions#1[#2]#3{%
  \@ifnextchar[%]
  {\@fileswith@pti@ns#1[{#2}]#3}%
  {\@fileswith@pti@ns#1[{#2}]#3[]}}
\@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.

```
⟨/2ekernel⟩
⟨latexrelease⟩\IncludeInRelease{2017/01/01}%
⟨latexrelease⟩        {\@fileswith@pti@ns}{ifx tests in \@fileswith@pti@ns}%
```

```
⟨*2ekernel | latexrelease⟩
\def\@fileswith@pti@ns#1[#2]#3[#4]{%
  \ifx#1\@clsextension
    \ifx\@classoptionslist\relax
      \xdef\@classoptionslist{\zap@space#2 \@empty}%
      \def\reserved@a{%
        \@onefilewithoptions#3[{#2}][{#4}]#1%
        \@documentclasshook}%
    \else
      \def\reserved@a{%
        \@onefilewithoptions#3[{#2}][{#4}]#1}%
    \fi
  \else
```

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

```
    \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)

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

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

```
        \ifx\@nnil##1\@nnil\else
          \noexpand\@onefilewithoptions##1[{#2}][{#4}]%
          \noexpand\@pkgextension
        \fi
        \expandafter\reserved@b
      \fi}%
      \edef\reserved@a{\zap@space#3 \@empty}%
      \edef\reserved@a{\expandafter\reserved@b\reserved@a,\@nnil,}%
  \fi
  \reserved@a}
⟨/2ekernel | latexrelease⟩

⟨latexrelease⟩\EndIncludeInRelease
⟨latexrelease⟩\IncludeInRelease{0000/00/00}%
⟨latexrelease⟩         {\@fileswith@pti@ns}{ifx tests in \@fileswith@pti@ns}%
⟨latexrelease⟩\def\@fileswith@pti@ns#1[#2]#3[#4]{%
⟨latexrelease⟩   \ifx#1\@clsextension
⟨latexrelease⟩     \ifx\@classoptionslist\relax
⟨latexrelease⟩       \xdef\@classoptionslist{\zap@space#2 \@empty}%
⟨latexrelease⟩       \def\reserved@a{%
⟨latexrelease⟩         \@onefilewithoptions#3[{#2}][{#4}]#1%
⟨latexrelease⟩         \@documentclasshook}%
⟨latexrelease⟩     \else
⟨latexrelease⟩       \def\reserved@a{%
⟨latexrelease⟩         \@onefilewithoptions#3[{#2}][{#4}]#1}%
⟨latexrelease⟩     \fi
⟨latexrelease⟩   \else
⟨latexrelease⟩     \def\reserved@b##1,{%
⟨latexrelease⟩       \ifx\@nil##1\relax\else
⟨latexrelease⟩         \ifx\relax##1\relax\else
```

```
⟨latexrelease⟩          \noexpand\@onefilewithoptions##1[{#2}][{#4}]%
⟨latexrelease⟩          \noexpand\@pkgextension
⟨latexrelease⟩        \fi
⟨latexrelease⟩        \expandafter\reserved@b
⟨latexrelease⟩      \fi}%
⟨latexrelease⟩      \edef\reserved@a{\zap@space#3 \@empty}%
⟨latexrelease⟩      \edef\reserved@a{%
⟨latexrelease⟩        \expandafter\reserved@b\reserved@a,\@nil,}%
⟨latexrelease⟩  \fi
⟨latexrelease⟩  \reserved@a}
⟨latexrelease⟩\EndIncludeInRelease
⟨∗2ekernel⟩
\@onlypreamble\@fileswith@pti@ns
```

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

```
\def\@onefilewithoptions#1[#2][#3]#4{%
  \@pushfilename
  \xdef\@currname{#1}%
  \global\let\@currext#4%
  \expandafter\let\csname\@currname.\@currext-h@@k\endcsname\@empty
  \let\CurrentOption\@empty
  \@reset@ptions
  \makeatletter
```

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

```
  \def\reserved@a{%
    \@ifl@aded\@currext{#1}%
      {\@if@ptions\@currext{#1}{#2}{}%
        {\@latex@error
            {Option clash for \@cls@pkg\space #1}%
            {The package #1 has already been loaded
             with options:\MessageBreak
             \space\space[\@ptionlist{#1.\@currext}]\MessageBreak
             There has now been an attempt to load it
              with options\MessageBreak
             \space\space[#2]\MessageBreak
             Adding the global options:\MessageBreak
             \space\space
                  \@ptionlist{#1.\@currext},#2\MessageBreak
             to your \noexpand\documentclass declaration may fix this.%
             \MessageBreak
             Try typing \space <return> \space to proceed.}}}%
      {\@pass@ptions\@currext{#2}{#1}%
       \global\expandafter
       \let\csname ver@\@currname.\@currext\endcsname\@empty
       \InputIfFileExists
         {\@currname.\@currext}%
         {}%
         {\@missingfileerror\@currname\@currext}%
```

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

```
    \let\@unprocessedoptions\@@unprocessedoptions
```

```
    \csname\@currname.\@currext-h@@k\endcsname
    \expandafter\let\csname\@currname.\@currext-h@@k\endcsname
              \@undefined
    \@unprocessedoptions}%
    \@ifl@ter\@currext{#1}{#3}{}%
      {\@latex@warning@no@line
         {You have requested,\on@line,
          version\MessageBreak
            `#3' of \@cls@pkg\space #1,\MessageBreak
          but only version\MessageBreak
           `\csname ver@#1.\@currext\endcsname'\MessageBreak
          is available}}%
    \ifx\@currext\@clsextension\let\LoadClass\@twoloadclasserror\fi
    \@popfilename
    \@reset@ptions}%
  \reserved@a}
\@onlypreamble\@onefilewithoptions
```

`\@@fileswith@pti@ns` Save the definition (for error checking).

```
\let\@@fileswith@pti@ns\@fileswith@pti@ns
\@onlypreamble\@@fileswith@pti@ns
```

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

```
\def\@reset@ptions{%
  \global\ifx\@currext\@clsextension
    \let\default@ds\OptionNotUsed
   \else
    \let\default@ds\@unknownoptionerror
  \fi
  \global\let\ds@\@empty
  \global\let\@declaredoptions\@empty}
\@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` `\@enddocumenthook` Stuff to appear at the beginning or end of the document.

```
\ifx\@begindocumenthook\@undefined
  \let\@begindocumenthook\@empty
\fi
\let\@enddocumenthook\@empty
```

`\g@addto@macro` Globally add to the end of a macro.

```
\long\def\g@addto@macro#1#2{%
  \begingroup
    \toks@\expandafter{#1#2}%
    \xdef#1{\the\toks@}%
  \endgroup}
```

\AtEndOfPackage
\AtEndOfClass
\AtBeginDocument
\AtEndDocument

The access functions.

```
\def\AtEndOfPackage{%
  \expandafter\g@addto@macro\csname\@currname.\@currext-h@@k\endcsname}
\let\AtEndOfClass\AtEndOfPackage
\@onlypreamble\AtEndOfPackage
\@onlypreamble\AtEndOfClass

\def\AtBeginDocument{\g@addto@macro\@begindocumenthook}
\def\AtEndDocument{\g@addto@macro\@enddocumenthook}
\@onlypreamble\AtBeginDocument
```

\@cls@pkg

The current file type.

```
\def\@cls@pkg{%
  \ifx\@currext\@clsextension
    document class%
  \else
    package%
  \fi}
\@onlypreamble\@cls@pkg
```

\@unknownoptionerror

Bad option.

```
\def\@unknownoptionerror{%
  \@latex@error
    {Unknown option `\CurrentOption' for \@cls@pkg\space`\@currname'}%
    {The option `\CurrentOption' was not declared in
     \@cls@pkg\space`\@currname', perhaps you\MessageBreak
      misspelled its name.
     Try typing \space <return>
     \space to proceed.}}
\@onlypreamble\@unknownoptionerror
```

\@@unprocessedoptions

Declare an error for each option, unless a \ProcessOptions occurred.

```
\def\@@unprocessedoptions{%
  \ifx\@currext\@pkgextension
    \edef\@curroptions{\@ptionlist{\@currname.\@currext}}%
    \@for\CurrentOption:=\@curroptions\do{%
        \ifx\CurrentOption\@empty\else\@unknownoptionerror\fi}%
  \fi}
\@onlypreamble\@unprocessedoptions
\@onlypreamble\@@unprocessedoptions
```

\@badrequireerror

\RequirePackage or \LoadClass occurs in the options section.

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

\@twoloadclasserror

Two \LoadClass in a class.

```
\def\@twoloadclasserror{%
  \@latex@error
```

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

`\@twoclasseserror` Two `\documentclass` or `\documentstyle`.

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

4.2 Providing shipment

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

```
\def\two@digits#1{\ifnum#1<10 0\fi\number#1}
```

`\filecontents` `\endfilecontents` This environment implements inline files. The star-form does not write extra comments into the file.

```
\begingroup%
\@tempcnta=1
\loop
  \catcode\@tempcnta=12  %
  \advance\@tempcnta\@ne %
\ifnum\@tempcnta<32      %
\repeat                  %
\catcode`\*=11 %
\catcode`\^^M\active%
\catcode`\^^L\active\let^^L\relax%
\catcode`\^^I\active%

\gdef\filecontents{\@tempswatrue\filec@ntents}%
\gdef\filecontents*{\@tempswafalse\filec@ntents}%

\gdef\filec@ntents#1{%
  \openin\@inputcheck#1 %
  \ifeof\@inputcheck%
    \@latex@warning@no@line%
        {Writing file `\@currdir#1'}%

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

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

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

```
      \@percentchar\@percentchar\space  generated by the %
        `\@currenvir' \expandafter\@gobblefour\string\newenvironment^^J%
      \@percentchar\@percentchar\space from source `\jobname' on %
         \number\year/\two@digits\month/\two@digits\day.^^J%
      \@percentchar\@percentchar}%
  \fi%
  \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.

```
  \count@ 128\relax%
  \loop%
    \catcode\count@ 11\relax%
    \advance\count@ \@ne%
    \ifnum\count@<\@cclvi%
  \repeat%
  \edef\E{\@backslashchar end\string{\@currenvir\string}}%
  \edef\reserved@b{%
    \def\noexpand\reserved@b%
         ####1\E####2\E####3\relax}%
  \reserved@b{%
    \ifx\relax##3\relax%
```

There was no `\end{filecontents}`

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

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

```
      \edef^^M{\noexpand\end{\@currenvir}}%
      \ifx\relax##1\relax%
      \else%
```

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

```
          \@latex@warning{Writing text `##1' before %
             \string\end{\@currenvir}\MessageBreak as last line of #1}%
        \immediate\write\reserved@c{##1}%
      \fi%
      \ifx\relax##2\relax%
      \else%
```

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

```
         \@latex@warning{%
           Ignoring text `##2' after \string\end{\@currenvir}}%
      \fi%
    \fi%
    ^^M}%
  \catcode`\^^L\active%
  \let\L\@undefined%
  \def^^L{\@ifundefined L^^J^^J^^J}%
  \catcode`\^^I\active%
  \let\I\@undefined%
  \def^^I{\@ifundefined I\space\space}%
  \catcode`\^^M\active%
  \edef^^M##1^^M{%
    \noexpand\reserved@b##1\E\E\relax}}%
\endgroup%
```

```
\begingroup
\catcode`|=\catcode`\%
\catcode`\%=12
\catcode`\*=11
\gdef\@percentchar{%}
\gdef\endfilecontents{|
  \immediate\closeout\reserved@c
  \def\T##1##2##3{|
  \ifx##1\@undefined\else
    \@latex@warning@no@line{##2 has been converted to Blank ##3e}|
  \fi}|
  \T\L{Form Feed}{Lin}|
  \T\I{Tab}{Spac}|
  \immediate\write\@unused{}}
\global\let\endfilecontents*\endfilecontents
\@onlypreamble\filecontents
\@onlypreamble\endfilecontents
\@onlypreamble\filecontents*
\@onlypreamble\endfilecontents*
\endgroup
\@onlypreamble\filec@ntents
```

5 Package/class rollback mechanism

```
⟨/2ekernel⟩
⟨*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.

```
⟨*tracerollback⟩
%\let\pkgcls@debug\typeout
\let\pkgcls@debug\@gobble
⟨/tracerollback⟩
```

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

```
\def\requestedLaTeXdate{0}
```

`\pkgcls@targetdate` `\pkgcls@targetlabel` `\pkgcls@innerdate` 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 verion 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.

```
\ifx\pkgcls@targetdate\@undefined
```

```
  \newcount\pkgcls@targetdate
\fi
\let\pkgcls@targetlabel\@empty
\def\pkgcls@innerdate{\maxdimen}
```

`\pkgcls@candidate` `\pkgcls@releasedate`

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.

```
\let\pkgcls@candidate\@empty
\let\pkgcls@releasedate\@empty
```

`\load@onefilewithoptions` `\@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.

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

First a bit of tracing normally disabled.

```
⟨*tracerollback⟩
  \pkgcls@debug{--- File loaded request (\noexpand\usepackage or ...)}%
  \pkgcls@debug{\@spaces 1: #1}%
  \pkgcls@debug{\@spaces 2: #2}%
  \pkgcls@debug{\@spaces 3: #3}%
  \pkgcls@debug{\@spaces 4: #4}%
⟨/tracerollback⟩
```

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

```
  \def\pkgcls@name{#1}%                  % for info message
  \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`.

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

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

```
  \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`.

```
  \load@onefilewithoptions#1[#2][\pkgcls@mindate]#4%
 }
\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 tht 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		`\pkgcls@targetdate`	`\pkgcls@targetlabel`	`\pkgcls@mindate`
⟨*empty*⟩	→	⟨*global-rollbackdate-as-number*⟩	⟨*empty*⟩	⟨*empty*⟩
⟨*date*⟩	→	⟨*global-rollbackdate-as-number*⟩	⟨*empty*⟩	⟨*date*⟩
`=`⟨*date*⟩	→	⟨*date-as-number*⟩	⟨*empty*⟩	⟨*empty*⟩
`=`⟨*version*⟩	→	`1`	⟨*version*⟩	⟨*empty*⟩
⟨*other*⟩	→	⟨*global-rollbackdate-as-number*⟩	⟨*empty*⟩	⟨*other*⟩

where ⟨*global-rollbackdate-as-number*⟩ is a date request given via `latexrelease` or if there wasn't one `0`.

```
\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.

```
  \ifx\@nil#1\@nil
    \pkgcls@targetdate\requestedLaTeXdate\relax
    \let\pkgcls@targetlabel\@empty
    \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.

```
  \else
    \pkgcls@parse@date@arg@#1=\@nil\relax
  \fi
}
```

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

```
\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.

```
  \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.

```
    \ifx\@nil#1\@nil
      \@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 `=`.

```
      \ifnum \pkgcls@targetdate=\z@
        \pkgcls@targetdate\@ne
        \def\pkgcls@innerdate{\maxdimen}%
        \pkgcls@parse@date@arg@version#2%
      \else
        \edef\pkgcls@innerdate{\the\pkgcls@targetdate}%
      \fi
      \let\pkgcls@mindate\@empty
    \else
```

If `#1` was not empty then there wasn't a `=` character in first position so we 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`):

```
      \requestedLaTeXdate\relax
      \let\pkgcls@targetlabel\@empty
      \def\pkgcls@innerdate{\maxdimen}%
      \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.

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

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

```
\def\pkgcls@parse@date@arg@version#1={%
  \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.

```
\def\DeclareRelease#1#2#3{%
  \ifnum\pkgcls@targetdate>\z@  % some sort of rollback request
⟨*tracerollback⟩
    \pkgcls@debug{---\string\DeclareRelease:}%
    \pkgcls@debug{\@spaces 1: #1}%
    \pkgcls@debug{\@spaces 2: #2}%
    \pkgcls@debug{\@spaces 3: #3}%
⟨/tracerollback⟩
```

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.

```
    \ifx\@nil#2\@nil
      \ifnum\pkgcls@targetdate=\@ne  % named request
        \def\reserved@a{#1}%
        \ifx\pkgcls@targetlabel\reserved@a
          \pkgcls@use@this@release{#3}{}%
⟨*tracerollback⟩
        \else
          \pkgcls@debug{Label doesn't match}%
⟨/tracerollback⟩
        \fi
⟨*tracerollback⟩
      \else
        \pkgcls@debug{Date request: ignored}%
⟨/tracerollback⟩
      \fi
    \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.

```
      \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.

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

```
          \ifx\pkgcls@candidate\@empty
            \pkgcls@rollbackdate@error{#2}%
            \pkgcls@use@this@release{#3}{#2}%
          \else
            \pkgcls@use@this@release\pkgcls@candidate
                                    \pkgcls@releasedate
          \fi
        \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.

```
            \def\pkgcls@candidate{#3}%
            \def\pkgcls@releasedate{#2}%
⟨*tracerollback⟩
            \pkgcls@debug{New candidate: #3}%
⟨/tracerollback⟩
          \fi
        \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.

```
          \def\reserved@a{#1}%
          \ifx\pkgcls@targetlabel\reserved@a
            \pkgcls@use@this@release{#3}{#2}%
⟨*tracerollback⟩
          \else
            \pkgcls@debug{Label doesn't match}%
⟨/tracerollback⟩
          \fi
        \fi
      \fi
    \fi
}
```

`\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.

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

Before that we record the selection made inside the transcript.

```
  \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 bypassed[1] 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`!

```
  \pkgcls@targetdate\z@
  \@@input #1\relax
  \endinput
}
```

`\pkgcls@show@selection` This command records what selection was made. As that is needed in two places (and it is rather lengthly) 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

[1] The older release may also have such declarations inside if it was a simply copy od 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.

```
\def\pkgcls@show@selection#1#2{%
⟨*tracerollback⟩
  \pkgcls@debug{Result: use  #1}%
⟨/tracerollback⟩
  \GenericInfo
   {\@spaces\@spaces\space}{Rollback for
    \@cls@pkg\space'\@currname' requested ->
    \ifnum\pkgcls@targetdate>\@ne
       date
       \ifnum\requestedLaTeXdate=\pkgcls@targetdate
          \requestedpatchdate
       \else
          \expandafter\@gobble\pkgcls@arg
       \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.

```
       Best approximation is
    \else
       version '\pkgcls@targetlabel'.\MessageBreak
       This corresponds to
    \fi
    \ifx\@nil#2\@nil
       a special release%
    \else
       the release introduced on #2%
    \fi
    \@gobble}%
}
```

`\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 is inlined.

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

```
\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).

```
  \ifnum\pkgcls@targetdate>\z@  % some sort of rollback request
⟨*tracerollback⟩
```

```
    \pkgcls@debug{---DeclareCurrentRelease}%
    \pkgcls@debug{   1: #1}%
    \pkgcls@debug{   2: #2}%
⟨/tracerollback⟩
```

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`.

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

```

        \ifx\pkgcls@candidate\@empty
          \pkgcls@rollbackdate@error{#2}%
        \else
          \pkgcls@use@this@release\pkgcls@candidate
                                   \pkgcls@releasedate
        \fi
```

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

```
      \else
        \pkgcls@show@selection{current version}{#2}%
      \fi
    \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.

```
      \def\reserved@a{#1}%
      \ifx\pkgcls@targetlabel\reserved@a
        \pkgcls@show@selection{current version}{#2}%
      \else
        \@latex@error{Requested version '\pkgcls@targetlabel' for
          \@cls@pkg\space'\@currname' is unknown}\@ehc
      \fi
    \fi
  \fi
}
```

`\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.

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

```
    \typeout{Exclude code introduced on #1}%
    \expandafter\@firstoftwo
  \else
    \typeout{Include code introduced on #1}%
    \expandafter\@secondoftwo
  \fi
}
```

⟨/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}`.

⟨∗afterpreamble⟩

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{pkgindoc}
          [1994/10/20 v1.1 Package Interface in Document (DPC)]
\def\reserved@a#1\do\@classoptionslist#2\do\filec@ntents#3\relax{%
  \gdef\@preamblecmds{#1#3}}
\expandafter\reserved@a\@preamblecmds\relax
```

⟨/afterpreamble⟩