The filemod Package

Martin Scharrer
martin@scharrer-online.de

http://www.ctan.org/pkg/filemod

Version v1.1 – 2011/03/24

Abstract

This package provides macros to read and compare the modification dates of files. These files can be `.tex` files, images or other files as long as they can be found by the LaTeX compiler. It uses the `\pdffilemoddate` primitive of pdfLaTeX to receive the file modification date as PDF date string, parses it and returns the value to the user. The functionality is provided by purely expandable macros or by faster but non-expandable ones.

This package will work with LaTeX and plain ε-TeX as long pdf(La)TeX (in PDF or DVI mode) or Lua(La)TeX is used. XeTeX is not supported because it does not provide `\pdffilemoddate`.

Contents

1 Introduction **1**

2 Usage **2**
2.1 Print File Modification Date and Time 2
2.2 Get File Modification Date and Time as Number 3
2.3 Compare File Modification Date/Time 4
2.4 Return Newest or Oldest File from a List 5
2.5 Parsing of the file modification date 6
2.6 Auxiliary Macros 6

3 Implementation **8**
3.1 Parser 8
3.2 Minimal set of expandable Macros 10
3.3 Expandable Macros . . . 12
3.4 Non-Expandable Macros 22
3.5 Display Macros 27
3.6 Auxiliary Macros 29

1 Introduction

This package provides several macros to read and compare the modification dates of files. The same functionality is provided by two groups of macros: The macros of the first group all start with a lower case letter and are fully expandable. This means they can be used in places where a string must be provided, like in `\input`. Because assignments are not expandable some of these macros, like the ones for comparisons, need to reread and re-parse the file modification dates if they are required in more than one place inside the macro. The macros of the second group all start with a upper case letter and are

not expandable because assignments are used internally. However, this allows techniques which speed up the processing of these macros, making this macros faster than the expandable counterparts. If expandability is not required these macros should be preferred.

2 Usage

This package can be loaded with LaTeX using `\usepackage{filemod}` as usual. With plain ε-TeX it can be loaded using `\input filemod`. Some required internal LaTeX macros (like `\@gobble`, `\@firstofone`, etc.) are then defined.

A minimal set of expandable macros for the comparison of file modification dates is provided by the sub-package `filemod-expmin`. It is useful for other packages which need this functionality but don't like to load the whole package. It can be loaded using `\usepackage{filemod-expmin}` (or `\RequirePackage`) or `\input filemod-expmin`, respectively.

2.1 Print File Modification Date and Time

The following macros can be used to print (i.e. typeset) the file modification date and time of files in the document. The `\formatdate` and `\formattime` macros of the `datetime`[1] can be used in addition to format the dates and times in a language specific format. See also the `getfiledate`[2] package which also prints file modification dates including adding fancy frames around it.

`\filemodprint{`⟨*filename*⟩`}`

Prints the file modifications date and time using `\filemodparse` and `\thefilemod`.

`\filemodprintdate{`⟨*filename*⟩`}`

Prints the file modifications date using `\filemodparse` and `\thefilemoddate`.

`\filemodprinttime{`⟨*filename*⟩`}`

Prints the file modifications time using `\filemodparse` and `\thefilemodtime`.

`\thefilemod`

Reads the date and time as seven arguments and typesets it. This macro can be redefined to a custom format.
By default it simple uses `\thefilemoddate` and `\thefilemodtime` separated by `\filemodsep` (a space by default): "2011/03/24 19:21:01 Z"

[1] CTAN: http://www.ctan.org/pkg/datetime
[2] CTAN: http://www.ctan.org/pkg/getfiledate

`\thefilemoddate`

Receives the date as three arguments YYYY, MM and DD and typesets it. This macro can be redefined to a custom format.
Default format: "2011/03/24"

It could be redefined to use the `\formatdate` macro of the `datetime`:

```
\renewcommand*{\thefilemoddate}[3]{\formatdate{#3}{#2}{#1}}
```

`\thefilemodtime`

Receives the time and timezone as four arguments HH, mm, SS and TZ and typesets it. This macro can be redefined to a custom format.
Default format: "19:21:01 Z"

It could be redefined to use the `\formattime` macro of the `datetime`:

```
\renewcommand*{\thefilemodtime}[4]{\formattime{#1}{#2}{#3}}
```

2.2 Get File Modification Date and Time as Number

The following macros return both the file modification date and time as an integer number which is in the valid range for TEX. They can be used for numerical operations and are used internally by the comparison macros.

`\filemodnumdate{`*⟨filename⟩*`}`

Expands to an integer of the form `YYYYMMDD` which can be used for numeric comparisons like `\ifnum`. This macros uses `\filemodparse` and `\filemodnotexists` will be used if the file does not exist.

`\filemodnumtime{`*⟨filename⟩*`}`

Expands to an integer of the form `HHmmSS` which can be used for numeric comparisons like `\ifnum`. This macros uses `\filemodparse` and `\filemodnotexists` will be used if the file does not exist.

`\filemodNumdate{`*⟨filename⟩*`}`

Expands to an integer of the form `YYYYMMDD` which can be used for numeric comparisons like `\ifnum`. Parses the file modification date by itself and will return `00000000` if the file does not exist.

`\filemodNumtime{`*⟨filename⟩*`}`

Expands to an integer of the form `HHmmSS` which can be used for numeric comparisons like `\ifnum`. Parses the file modification date by itself and will return `000000` if the file does not exist.

`\Filemodgetnum{`*⟨filename⟩*`}`

Stores the file modification date and time as numbers (`YYYYMMDD` and `HHmmSS`) as well the timezone string into the macros `\filemoddate`, `\filemodtime` and `\filemodtz`.

2.3 Compare File Modification Date/Time

The following macros allow the comparison of the file modification date/time of two files.

`\filemodcmp`[⟨*num*⟩]{⟨*filename 1*⟩}{⟨*filename 2*⟩}{⟨*clause 1*⟩}{⟨*clause 2*⟩}{⟨*clause 3*⟩}

This macro compares the file modification date and time of the two given files and expands to the clause of the newest file. An numerical optional argument can be given to determine the outcome if both files have the exact same modification date/time (or both do not exists). If ⟨*num*⟩ is 0, no clause will be expanded, i.e. the macro expands to an empty text. If ⟨*num*⟩ is 1 (default) or 2 the macro expands to the corresponding clause. However if ⟨*num*⟩ is 3, the macro will await a third clause and expands to it if both files modification dates are equal.

This macro is fully expandable even when the optional argument is used. However, ⟨*filename 1*⟩ must not be equal to '`[`'.

`\filemodCmp`{⟨*filename 1*⟩}{⟨*filename 2*⟩}{⟨*clause 1*⟩}{⟨*clause 2*⟩}

This is a simpler and therefore faster version of `\filemodcmp`. It is fully expandable, does not take any optional arguments and will always expand to the first clause if both file modification dates are equal (or both files do not exist). The `\filemodNumdate` and `\filemodNumtime` macros are used in the comparison. These three macros are also provided by the sub-package `filemod-expmin`.

`\Filemodcmp`[⟨*num*⟩]{⟨*filename 1*⟩}{⟨*filename 2*⟩}{⟨*clause 1*⟩}{⟨*clause 2*⟩}{⟨*clause 3*⟩}

This macro provides the same functionality as `\filemodcmp`. It is not expandable but will be processed faster. The optional argument is processed like normally.

`\FilemodCmp`[⟨*num*⟩]{⟨*filename 1*⟩}{⟨*filename 2*⟩}

This macro will compare the two file modification dates like `\Filemodcmp` and `\filemodcmp` but does not take the possible clauses as arguments, instead it stores the result into the expandable macro `\filemodcmpresult` which then takes {⟨*clause 1*⟩}{⟨*clause 2*⟩} (and also {⟨*clause 3*⟩} if ⟨*num*⟩ was 3) as arguments and expand to the one corresponding to the newest file. This set of macros gives the user the speed benefit of `\Filemodcmp` while still be able to use the result in an expandable context.

`\filemodoptdefault`

Holds the default number (i.e. 1) for the optional argument of the previous and following macros. This macro can be redefined with a number or a numeric expression valid for `\ifcase`. It should not contain any trailing spaces. Note that some commands only accept 1 or 2 as valid optional argument.

2.4 Return Newest or Oldest File from a List

The following macros return the newest or oldest file. Note that the optional arguments of the following macros should only be either 1 or 2. If no optional argument is provided the value of `\filemodoptdefault` is used.

`\filemodnewest`[⟨*num*⟩]{⟨*filename 1*⟩}{⟨*filename 2*⟩}

Expands the filename of the newest given file or filename ⟨*num*⟩ if both file modification dates are identical. The catcode of the filenames is not changed.

`\filemodoldest`[⟨*num*⟩]{⟨*filename 1*⟩}{⟨*filename 2*⟩}

Expands the filename of the oldest given file or filename ⟨*num*⟩ if both file modification dates are identical. The catcode of the filenames is not changed.

`\filemodNewest`[⟨*num*⟩]{{⟨*filename 1*⟩}{⟨*filename 2*⟩}...{⟨*filename n*⟩}}

Expands the filename of the newest given file. The filename will have catcode 12 except in the case when only one filename was given which is returned unchanged. The files are compared in pairs of two in the given order (i.e. first 1 and 2 and the result with 3 etc.) The optional argument ⟨*num*⟩ can be used to indicate which filename should be used if both file modification dates are identical.

`\filemodOldest`[⟨*num*⟩]{{⟨*filename 1*⟩}{⟨*filename 2*⟩}...{⟨*filename n*⟩}}

Expands the filename of the oldest given file. The filename will have catcode 12 except in the case when only one filename was given which is returned unchanged. The files are compared in pairs of two in the given order (i.e. first 1 and 2 and the result with 3 etc.) The optional argument ⟨*num*⟩ can be used to indicate which filename should be used if both file modification dates are identical.

`\Filemodnewest`[⟨*num*⟩]{⟨*filename 1*⟩}{⟨*filename 2*⟩}

Same as `\filemodnewest` just not expandable but faster. Stores the newer of the two file names in `\filemodresultfile`. Its file modification date and time is stored in `\filemodresultdate` and `\filemodresulttime`. The catcode of the filenames is not changed.

`\Filemodoldest`[⟨*num*⟩]{⟨*filename 1*⟩}{⟨*filename 2*⟩}

Same as `\filemodoldest` just not expandable but faster. Stores the older of the two file names in `\filemodresultfile`. Its file modification date and time is stored in `\filemodresultdate` and `\filemodresulttime`. The catcode of the filenames is not changed.

`\FilemodNewest[⟨num⟩]{{⟨filename 1⟩}{⟨filename 2⟩}...{⟨filename n⟩}}`

Same as `\filemodNewest` just not expandable but faster. Stores the newest of the given file names in `\filemodresultfile`. Its file modification date and time is stored in `\filemodresultdate` and `\filemodresulttime`. The catcode of the filenames is not changed.

`\FilemodOldest[⟨num⟩]{{⟨filename 1⟩}{⟨filename 2⟩}...{⟨filename n⟩}}`

Same as `\filemodOldest` just not expandable but faster. Stores the oldest of the given file names in `\filemodresultfile`. Its file modification date and time is stored in `\filemodresultdate` and `\filemodresulttime`. The catcode of the filenames is not changed.

2.5 Parsing of the file modification date

The format returned by the `\pdffilemoddate` primitive is "`D:`" followed by a number in the format "`YYYYMMDDHHmmSST`" which needs to be parsed before it is useful. The letters have the following meaning: `Y` = year, `M` = month, `D` = day, `H` = hour, `mm` = minutes, `S` = seconds, `T` or `TZ` = timezone string. The number of letters indicates the length except for the timezone which is of variable length. An example is "`D:20110324192101Z`" which is the file modification date of the source file of this manual. Unfortunately this number is to large for TeX to be taken as an integer for numerical comparisons, so it is broken into two numbers (`YYYYMMDD` and `HHmmSS`) which are compared in multiple steps.

`\filemodparse{⟨macro⟩}{⟨filename⟩}`

Parses the file modification datetime of the given file and passes the result to the given macro. The macro will receive seven arguments:

⟨*macro*⟩{⟨*YYYY*⟩}{⟨*MM*⟩}{⟨*DD*⟩}{⟨*HH*⟩}{⟨*mm*⟩}{⟨*SS*⟩}{⟨*TZ*⟩}

i.e. year, month, day, hour, minutes, seconds and the timezone as signed offset or Z (catcode 12).

`\filemodnotexists{⟨macro⟩}`

This macro will be called by `\filemodparse` with the original given macro when the given file does not exists. By default it contains all zeros except Z (catcode 12) as timezone:

```
#1{0000}{00}{00}{00}{00}{00}{Z}
```

The user can redefine this macro to a different content, e.g. to a different fall-back value or to display a warning. Note if this macro contains non-expandable code the macros which uses it aren't expandable anymore.

2.6 Auxiliary Macros

`\filemodZ`

Defined to 'Z' with catcode 12 as it is returned as timezone. This might be useful for comparisons or custom definitions.

`\filemodz`

Let (`\let`) to 'Z' with catcode 12 as it is returned as timezone. This might be useful for comparisons or custom definitions.

3 Implementation

Ensure correct catcode for plainTeX:

```
%<*tex>
\expandafter\edef\csname filemod@cat\endcsname{\
   noexpand\catcode`\noexpand\@=\the\catcode`\@\relax
   }
\catcode`\@=11
%</tex>
```

Check if the `\pdffilemoddate` command is available. If not (e.g. with LuaLaTeX) the `pdftexcmds` is loaded to provide the `\pdf@filemoddate` replacement. However for XeLaTeX this will fail and an error is raised.

```
\ifx\filemod@directtrue\@undefined
\csname newif\expandafter\endcsname\csname
   iffilemod@direct\endcsname
\filemod@directtrue
\ifx\pdffilemoddate\@undefined
%<latex> \RequirePackage{pdftexcmds}
%<tex>   \input pdftexcmds.sty
    \filemod@directfalse
    \ifx\pdf@filemoddate\@undefined
        \edef\filemod@help
        {The required command \string\pdffilemoddate\
           space is not defined.
         This means that the used\space\space LaTeX
            compiler does not support it.
         Please make sure that pdfLaTeX 1.30.0 or\
            space\space\space newer or LuaLaTeX is
            used.
         XeLaTeX does not support reading file
            modification\space\space dates.
        }%
%<latex>  \PackageError{filemod}{Required command \
   string\pdffilemoddate\space is not defined!}{\
   filemod@help}
%<tex>    \errhelp\expandafter{\filemod@help}
%<tex>    \errmessage{filemod package: Required
   command \string\pdffilemoddate\space is not
   defined!}
    \fi
\fi
\fi
```

3.1 Parser

\filemodparse

#1: Macro or tokens to process result
#2: file name

```
%<latex>\newcommand*\filemodparse{}
\iffilemod@direct
\def\filemodparse#1#2{%
    \expandafter\filemod@parse\pdffilemoddate{#2}\↙
       relax{#1}%
}
\else
\def\filemodparse#1#2{%
    \expandafter\expandafter
    \expandafter\filemod@parse\pdf@filemoddate{#2}\↙
       relax{#1}%
}
\fi
```

\filemod@parse

#1: Expanded file mod date
#2: Macro

```
\def\filemod@parse#1\relax#2{%
    \ifx\relax#1\relax
        \expandafter\@firstoftwo
    \else
        \expandafter\@secondoftwo
    \fi
    {\filemodnotexists{#2}}%
    {\filemod@parse@#1\empty{#2}\relax}%
}
```

The 'D', ':' and 'Z' characters are changed to catcode 12 because this is how they appear in the string returned by \pdffilemoddate.

```
\begingroup
\catcode`\D=12
\catcode`\Z=12
\catcode`\:=12
```

\filemod@parse@

#1: Y1
#2: Y2
#3: Y3
#4: Y4
#5: M1

#6: M2
#7: D1
#8: D2
#9: Rest

```
\gdef\filemod@parse@ D:#1#2#3#4#5#6#7#8#9\relax{%
    \filemod@parse@@{{#1#2#3#4}{#5#6}{#7#8}}#9\relax
}
```

\filemodnotexists

#1: Macro provided to \filemodparse
Macro which is used for non-existing files.

```
%<latex>\newcommand*\filemodnotexists{}
\gdef\filemodnotexists#1{%
    #1{0000}{00}{00}{00}{00}{00}{Z}%
}

\endgroup
```

\filemod@parse@@

#1: {YYYY}{MM}{DD}
#2: H1
#3: H2
#4: m1
#5: m2
#6: S1
#7: S2
#8: TZ
#9: Macro
Reads the rest of the file mod date and places the resulting arguments in front of the given macro.

```
\def\filemod@parse@@#1#2#3#4#5#6#7#8\empty#9\relax{%
    #9#1{#2#3}{#4#5}{#6#7}{#8}%
}
```

3.2 Minimal set of expandable Macros

The 'D', ':' and 'Z' characters are changed to catcode 12 because this is how they appear in the string returned by \pdffilemoddate.

```
\begingroup
\catcode`\D=12
\catcode`\Z=12
\catcode`\:=12
```

\filemodNumdate

```
%<latex>\newcommand*\filemodNumdate{}
\iffilemod@direct
\gdef\filemodNumdate#1{%
    \expandafter\filemod@Numdate\pdffilemoddate{#1}D:000000000000000Z\relax
}
\else
\gdef\filemodNumdate#1{%
    \expandafter\expandafter
    \expandafter\filemod@Numdate\pdf@filemoddate{#1}D:000000000000000Z\relax
}
\fi
```

\filemod@Numdate

```
\gdef\filemod@Numdate D:#1#2#3#4#5#6#7#8#9\relax{%
    #1#2#3#4#5#6#7#8%
}
```

\filemodNumtime

```
%<latex>\newcommand*\filemodNumtime{}
\iffilemod@direct
\gdef\filemodNumtime#1{%
    \expandafter\filemod@Numtime\pdffilemoddate{#1}D:000000000000000Z\relax
}
\else
\gdef\filemodNumtime#1{%
    \expandafter\expandafter
    \expandafter\filemod@Numtime\pdf@filemoddate{#1}D:000000000000000Z\relax
}
\fi
```

\filemod@Numtime

```
\gdef\filemod@Numtime D:#1#2#3#4#5#6#7#8#9\relax{%
    \filemod@@Numtime#9\relax
}
```

`\filemod@@Numtime`

```
\gdef\filemod@@Numtime#1#2#3#4#5#6#7\relax{%
    #1#2#3#4#5#6%
}

\endgroup
```

`\filemodCmp`

```
%<latex>\newcommand*\filemodCmp[2]%
%<tex>\def\filemodCmp#1#2%
{%
    \ifcase0%
        \ifnum\filemodNumdate{#2}>\filemodNumdate{#1}↙
            1\else
        \ifnum\filemodNumdate{#2}=\filemodNumdate↙
            {#1} %
             \ifnum\filemodNumtime{#2}>\↙
                filemodNumtime{#1} 1\fi
        \fi
    \fi
    \space
       \expandafter\@firstoftwo
    \or
       \expandafter\@secondoftwo
    \fi
}
```

Some required LaTeX macros for the plainTeX version:

```
%<*tex>
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
%</tex>

%<latex>\RequirePackage{filemod-expmin}
%<tex>\input filemod-expmin
```

3.3 Expandable Macros

3.3.1 Numeric macros

`\filemodnumdate`

Simply calls the parse macro.

```
%<latex>\newcommand*%
%<tex>\def
\filemodnumdate{\filemodparse\filemod@numdate}
```

\filemod@numdate

#1: YYYY
#2: MM
#3: DD
#4: HH
#5: mm
#6: SS
#7: TZ

```
% Gobbles everything except "YYYYMMDD" which is ↙
    returned as number without the braces.
\def\filemod@numdate#1#2#3#4#5#6#7{#1#2#3}
```

\filemodnumtime

Simply calls the parse macro.

```
%<latex>\newcommand*%
%<tex>\def
\filemodnumtime{\filemodparse\filemod@numtime}
```

\filemod@numtime

#1: YYYY
#2: MM
#3: DD
#4: HH
#5: mm
#6: SS
#7: TZ

Gobbles everything except 'HHmmSS' which is returned as number without the braces.

```
\def\filemod@numtime#1#2#3#4#5#6#7{#4#5#6}
```

3.3.2 Optional argument handler

\filemod@opt

#1: Macro to read optional argument when present
#2: Next macro which receives default optional argument as first normal argument
#3: [or first mandatory argument

This macro checks if an optional argument is present. Here #1 and #2 are handlers and #3 is the first balanced text which followed the macro, i.e. either '[' or the first mandatory argument. The \ifx compares '[' and the first token of #3. There are three possible cases:

1. If they do not match everything until and including \else is skipped. Then \remove@to@nnil@exec is expanded which removes the following \@nnil. This leaves \empty and the rest of the *false* clause. The \fi is removed using \expandafter and the trailing {#3} is read by #2 as normal argument.

2. If #3 is exactly '[' the \ifx[#3 part is removed by TEX. The \remove@to@nnil@exec removes the \@nnil and the \remove@to@nnil because there was nothing before \@nnil. Therefore \expandafter#1 is executed which triggers \else which removes everything up to and including \fi. Then the optional argument handler #1 is expanded which receives the '[' as '{[}' which is then gobbled.

3. The #3 starts with '[' but contains more material, i.e. was original a mandatory argument. Then \ifx expands to the *true* clause and removes the first token of #3. The \remove@to@nnil@exec gobbles the rest of #3 but reads and reinserts \remove@to@nnil which gobbles everything to the next \@nnil after \else and therefore jumps to the *false* clause. This clause is executed like normal, i.e. #2 is called with the default optional argument and {#3} as second argument.

```
\def\filemod@opt#1#2#3{%
    \expandafter
    \remove@to@nnil@exec
    \ifx[#3\@nnil\remove@to@nnil
      \expandafter#1%
    \else\@nnil\empty
      \expandafter#2%
      \expandafter\filemodoptdefault
    \fi
    {#3}%
}
```

\remove@to@nnil@exec

#1: Tokens to remove
#2: Following token

Removes everything to \@nnil and executes the next token except if #1 was empty.

```
\def\remove@to@nnil@exec#1\@nnil#2{%
    \ifx\@nnil#1\@nnil\else
      \expandafter#2
    \fi
}
```

3.3.3 Compare file dates

`\filemodcmp`

Compare two file mod dates. Calls macros to check for an optional argument in an expandable way.

```
%<latex>\newcommand*%
%<tex>\def
\filemodcmp{%
    \filemod@opt\filemod@cmp@opt\filemod@cmp
}
```

`\filemodoptdefault`

The default optional argument which is used if none is provided.

```
%<latex>\newcommand*%
%<tex>\def
\filemodoptdefault{1}
```

`\filemod@cmp@opt`

#1: '[' wrapped in {}
#2: Content of optional argument
Removes the brackets from the optional argument.

```
\def\filemod@cmp@opt#1#2]{%
    \filemod@cmp{#2}%
}
```

`\filemod@cmp`

This saves several `\expandafter`'s in `\filemod@opt`.

```
\def\filemod@cmp{\filemod@@cmp>}
```

`\filemod@@cmp`

#1: Compare sign: > or <
#2: Optional argument
#3: File name 1
#4: File name 2

Compares the dates and times of the two files. The three cases are (0) file 1 newer than file 2, (1) file 2 newer than file 1, (2) both files have the same date.

In (2) the optional argument #2 determines which clause is executed.

```
\def\filemod@@cmp#1#2#3#4{%
    \ifcase0%
        \ifnum\filemodnumdate{#4}#1\filemodnumdate{#3} 1\else
        \ifnum\filemodnumdate{#4}=\filemodnumdate{#3} %
            \ifnum\filemodnumtime{#4}#1\filemodnumtime{#3} 1\else
            \ifnum\filemodnumtime{#4}=\filemodnumtime{#3} 2\fi
            \fi
        \fi
    \fi
    \space
      \csname @firstoft\ifnum#2>2 hree\else wo\fi\expandafter\endcsname
    \or
      \csname @secondoft\ifnum#2>2 hree\else wo\fi\expandafter\endcsname
    \else
      \csname @%
      \ifcase#2%
        gobbletwo%
      \or
        firstoftwo%
      \or
        secondoftwo%
      \else
        thirdofthree%
      \fi
      \expandafter
      \endcsname
    \fi
}
```

`\@firstofthree`

Expands to the first of the next three arguments.

```
\long\def\@firstofthree#1#2#3{#1}
```

`\@secondofthree`

Expands to the second of the next three arguments.

```
\long\def\@secondofthree#1#2#3{#2}
```

Some required LaTeX macros for the plainTeX version:

```
%<*tex>
\long\def\@thirdofthree#1#2#3{#3}
\long\def\@gobble#1{}
\long\def\@gobbletwo#1#2{}
\def\remove@to@nnil#1\@nnil{}
%</tex>
```

3.3.4 Compare file mod times and return file name

`\filemodnewest`

First a macro is called to handle an optional argument in an expandable way.

```
%<latex>\newcommand*%
%<tex>\def
\filemodnewest{%
    \filemod@opt\filemod@newest@opt\filemod@newest
}
```

`\filemod@newest@opt`

#1: The '[' wrapped in {}
#2: Content of optional argument
Removes braces around the optional argument.

```
\def\filemod@newest@opt#1#2]{%
    \filemod@newest{#2}%
}
```

`\filemod@newest`

#1: optional argument
#2: file name 1
#3: file name 2
Uses the normal (internal) compare macro with the file names as the result clauses.

```
\def\filemod@newest#1#2#3{%
    \filemod@@cmp>{#1}{#2}{#3}{#2}{#3}%
}
```

`\filemodoldest`

First a macro is called to handle an optional argument in an expandable way.

```
%<latex>\newcommand*%
%<tex>\def
\filemodoldest{%
    \filemod@opt\filemod@oldest@opt\filemod@oldest
}
```

`\filemod@oldest@opt`

#1: The '[' wrapped in {}
#2: Content of optional argument
Removes braces around the optional argument.

```
\def\filemod@oldest@opt#1#2]{%
    \filemod@oldest{#2}%
}
```

`\filemod@oldest`

#1: optional argument
#2: file name 1
#3: file name 2
Uses the normal (internal) compare macro with the file names as the result clauses.

```
\def\filemod@oldest#1#2#3{%
    \filemod@@cmp<{#1}{#2}{#3}{#2}{#3}%
}
```

3.3.5 Newest and oldest file of a list of files

`\filemodNewest`

#1: Tokens between macros and opening brace
Checks for an optional argument and substitutes the default if it is missing.

```
%<latex>\newcommand*\filemodNewest{}
\def\filemodNewest#1#{%
  \expandafter\expandafter
  \expandafter\@filemodNewest
  \csname
    @%
  \ifx\@nnil#1\@nnil
    first%
  \else
    second%
  \fi
    oftwo%
  \endcsname
    {[\filemodoptdefault]}%
    {#1}%
}
```

\filemodOldest

#1: Tokens between macros and opening brace

Like \filemodNewest but returns the oldest file in the given list. It and its submacros are simply copies of minor changes of the Newest counterparts. This is done for the benefit of expansion speed versus memory usage. Future versions might use common code instead.

```
%<latex>\newcommand*\filemodOldest{}
\def\filemodOldest#1#{%
  \expandafter\expandafter
  \expandafter\@filemodOldest
  \csname
    @%
  \ifx\@nnil#1\@nnil
    first%
  \else
    second%
  \fi
    oftwo%
  \endcsname
    {[\filemodoptdefault]}%
    {#1}%
}
```

\@filemodNewest

#1: Optional argument

#2: File name list

Removes '[]' from first and braces from the second argument (the filename list).

```
\def\@filemodNewest[#1]#2{%
    \@@filemodNewest{#1}#2\filemod@end
}
```

\@filemodOldest

#1: Optional argument
#2: File name list
Like \@filemodNewest.

```
\def\@filemodOldest[#1]#2{%
    \@@filemodOldest{#1}#2\filemod@end
}
```

\@@filemodNewest

#1: Optional argument
#2: First file name
Reads the optional argument as #1 and the first filename as #2. It then reverses the order for the processing loop.

```
\def\@@filemodNewest#1#2{%
    \filemod@Newest{#2}{#1}%
}
```

\@@filemodOldest

#1: Optional argument
#2: First file name

```
\def\@@filemodOldest#1#2{%
    \filemod@Oldest{#2}{#1}%
}
```

\filemod@Newest

#1: First file name
#2: Optional argument
#3: Second file name
Checks if the second filename is the end marker. In this case the first filename is returned (i.e. expanded to). Otherwise expands the compare macro. This is done in one step using \csname which is then turned into a string which \ is gobbled. Because of the required expandability the \escapechar can't be changed. Finally it calls itself recursively with the expanded result.

```
\def\filemod@Newest#1#2#3{%
  \iffilemod@end{#3}%
    {#1}%
    {%
    \expandafter\expandafter
    \expandafter\expandafter
    \expandafter\expandafter
    \expandafter\filemod@Newest
    \expandafter\expandafter
    \expandafter\expandafter
    \expandafter\expandafter
    \expandafter{%
    \expandafter\expandafter
    \expandafter\@gobble
    \expandafter\string\csname\filemod@@cmp
       >{#2}{#1}{#3}{#1}{#3}\endcsname}{#2}}%
}
```

\filemod@Oldest

#1: First file name
#2: Optional argument
#3: Second file name

Like \filemode@Newest but with different compare operator.

```
\def\filemod@Oldest#1#2#3{%
  \iffilemod@end{#3}%
    {#1}%
    {%
    \expandafter\expandafter
    \expandafter\expandafter
    \expandafter\expandafter
    \expandafter\filemod@Oldest
    \expandafter\expandafter
    \expandafter\expandafter
    \expandafter\expandafter
    \expandafter{%
    \expandafter\expandafter
    \expandafter\@gobble
    \expandafter\string\csname\filemod@@cmp
       <{#2}{#1}{#3}{#1}{#3}\endcsname}{#2}}%
}
```

\iffilemod@end

#1: Next filename or end marker

Checks if the argument is the \filemod@end marker.

```
\def\iffilemod@end#1{%
  \ifx\filemod@end#1%
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
```

`\filemod@end`

Unique end marker which would expand to nothing. Could be replaced with `\@nnil`.

```
\def\filemod@end{\@gobble{filemod@end}}
```

3.4 Non-Expandable Macros

The following macros are not expandable but contain assignments which must be executed. This makes them faster because information can be buffered. Some of them can return expandable results.

3.4.1 Get Numeric Representation of File Modification Date

`\Filemodgetnum`

```
%<latex>\newcommand*%
%<tex>\def
\Filemodgetnum{\filemodparse\Filemod@getnum}
```

`\Filemod@getnum`

```
\def\Filemod@getnum#1#2#3#4#5#6#7{%
    \def\filemoddate{#1#2#3}%
    \def\filemodtime{#4#5#6}%
    \def\filemodtz{#7}%
}
```

3.4.2 Compare Two File Modification Dates

`\Filemodcmp`

`#1`: Optional argument (default: '1')
Calls `\Filemod@cmp` to execute the result at the end.

```
%<*latex>
\newcommand\Filemodcmp[1][1]{%
    \def\filemod@next{\filemodcmpresult}%
    \Filemod@cmp{#1}%
}
%</latex>
```

\FilemodCmp

Calls \Filemod@cmp to not execute the result at the end. Instead the user must use \filemodcmpresult explicitly.

```
%<*latex>
\newcommand\FilemodCmp[1][1]{%
    \let\filemod@next\empty
    \Filemod@cmp{#1}%
}
%</latex>
```

\Filemod@cmp

#1: Optional argument
#2: File name 1
#3: File name 2

Compares both files and defines \filemodcmpresult so that it expands to the winning clause. It might be directly executed at the end or not depending on the definition of \filemod@next which is set by the user level macros which use this macro.

```
\def\Filemod@cmp#1#2#3{%
    \Filemodgetnum{#2}%
    \let\filemoddatea\filemoddate
    \let\filemodtimea\filemodtime
    \Filemodgetnum{#3}%
    \ifcase0%
        \ifnum\filemoddate>\filemoddatea\space1\else
            \ifnum\filemoddate=\filemoddatea\space
                \ifnum\filemodtime>\filemodtimea\↙
                   space1\else
                    \ifnum\filemodtime=\filemodtimea\↙
                       space2\fi
                \fi
            \fi
        \fi
    \relax
```

First file is newer:

```
        \def\filemodresultfile{#1}%
        \ifnum#1>2\relax
           \def\filemodcmpresult##1##2##3{##1}%
        \else
           \let\filemodcmpresult\@firstoftwo
        \fi
    \or
```

Second file is newer:

```
        \def\filemodresultfile{#2}%
        \ifnum#1>2\relax
           \def\filemodcmpresult##1##2##3{##2}%
        \else
           \let\filemodcmpresult\@secondoftwo
        \fi
    \else
```

File mod dates are equal. The optional argument determines which clause is used.

```
        \ifcase#1\relax
          \let\filemodresultfile\empty
          \let\filemodcmpresult\@gobbletwo
        \or
          \def\filemodresultfile{#1}%
          \let\filemodcmpresult\@firstoftwo
        \or
          \def\filemodresultfile{#2}%
          \let\filemodcmpresult\@secondoftwo
        \else
          \let\filemodresultfile\empty
          \let\filemodcmpresult\@thirdofthree
        \fi
    \fi
    \filemod@next
}
```

`\filemodcmpresult`

Defined above.

3.4.3 Compare file mod times and return file name

Handlers for optional arguments for plainTEX. If none is provided the `\filemodoptdefault` is used.

```
%<*tex>
\def\filemod@chkopt#1{%
    \def\filemod@optcmd{#1}%
    \futurelet\filemod@tok\filemod@@chkopt
```

```
}
\def\filemod@@chkopt{%
    \ifx[\filemod@tok
        \expandafter\filemod@readopt
    \else
        \expandafter\filemod@optcmd
        \expandafter\filemodoptdefault
    \fi
}
\def\filemod@readopt[#1]{%
    \filemod@optcmd{#1}%
}
%</tex>
```

\Filemodnewest

Simply uses \FilemodNewest.

```
%<*latex>
\newcommand*\Filemodnewest[3][\filemodoptdefault]{\↙
   FilemodNewest[{#1}]{{#2}{#3}}}
%</latex>
%<*tex>
\def\Filemodnewest{\filemod@chkopt\Filemod@newest}
\def\Filemod@newest#1#2#3{\Filemod@Newest↙
   {#1}{{#2}{#3}}}
%</tex>
```

\Filemodoldest

Simply uses \FilemodOldest.

```
%<*latex>
\newcommand*\Filemodoldest[3][\filemodoptdefault]{\↙
   FilemodOldest[{#1}]{{#2}{#3}}}
%</latex>
%<*tex>
\def\Filemodoldest{\filemod@chkopt\Filemod@oldest}
\def\Filemod@oldest#1#2#3{\Filemod@Oldest↙
   {#1}{{#2}{#3}}}
%</tex>
```

\FilemodNewest

Uses \Filemod@est with a different compare sign. Stores the optional argument for later processing. This avoids the need to pass it around as an argument.

```
%<latex>\newcommand*\FilemodNewest[2][\↙
    filemodoptdefault]%
%<tex>\def\FilemodNewest{\filemod@chkopt\↙
    Filemod@Newest}
%<tex>\def\Filemod@Newest#1#2%
{%
     \def\filemode@tie{#1}%
     \def\filemod@gl{>}%
     \Filemod@est#2\filemod@end
}
```

\FilemodOldest

Uses \Filemod@est with a different compare sign. Stores the optional argument for later processing. This avoids the need to pass it around as an argument.

```
%<latex>\newcommand*\FilemodOldest[2][\↙
    filemodoptdefault]%
%<tex>\def\FilemodOldest{\filemod@chkopt\↙
    Filemod@Oldest}
%<tex>\def\Filemod@Oldest#1#2%
{%
     \def\filemode@tie{#1}%
     \def\filemod@gl{<}%
     \Filemod@est#2\filemod@end
}
```

\Filemod@est

#1: file name 1

Initiates the macros with the name, date and time of the first file. Then the recursive part is called.

```
\def\Filemod@est#1{%
     \def\filemodresultfile{#1}%
     \Filemodgetnum{#1}%
     \let\filemodresultdate\filemoddate
     \let\filemodresulttime\filemodtime
     \Filemod@@est
}
```

\Filemod@@est

#1: Next filename or end marker

Recursive part. Simple aborts (expands to nothing) if #1 is the end-marker. Then the resulting file is in \filemodresultfile and the date and time are in \filemodresultdate and \filemodresulttime, respectively.

```
\def\Filemod@@est#1{%
  \iffilemod@end{#1}{}{%
    \Filemodgetnum{#1}%
    \ifcase0%
        \ifnum\filemoddate\filemod@gl\↙
           filemodresultdate\space1\else
            \ifnum\filemoddate=\filemodresultdate\↙
               space
                \ifnum\filemodtime\filemod@gl\↙
                   filemodresulttime\space1\else
                    \ifnum\filemodtime=\↙
                       filemodresulttime\space
                        \ifnum\filemode@tie=1\else 1\↙
                           fi
                    \fi
                \fi
            \fi
        \fi
    \else
        \def\filemodresultfile{#1}%
        \let\filemodresultdate\filemoddate
        \let\filemodresulttime\filemodtime
    \fi
    \Filemod@@est
  }%
}
```

`\filemod@gl`

Initial value of compare sign. Not really required to be defined here because it is defined to the required sign every time it is used.

```
\def\filemod@gl{>}
```

3.5 Display Macros

`\filemodprint`

```
%<latex>\newcommand*
%<tex>\def
\filemodprint{\filemodparse\thefilemod}
```

`\filemodprintdate`

```
%<latex>\newcommand*
%<tex>\def
\filemodprintdate{\filemodparse\the@filemoddate}
```

\filemodprinttime

```
%<latex>\newcommand*
%<tex>\def
\filemodprinttime{\filemodparse\the@filemodtime}
```

\thefilemod

```
%<latex>\newcommand*\thefilemod[7]%
%<tex>\def\thefilemod#1#2#3#4#5#6#7%
{%
    \thefilemoddate{#1}{#2}{#3}%
    \filemodsep
    \thefilemodtime{#4}{#5}{#6}{#7}%
}

%<latex>\newcommand*\filemodsep{ }
%<tex>\let\filemodsep\space
```

\thefilemoddate

```
%<latex>\newcommand*\thefilemoddate[3]%
%<tex>\def\thefilemoddate#1#2#3%
{#1/#2/#3}
```

\thefilemodtime

```
%<latex>\newcommand*\thefilemodtime[4]%
%<tex>\def\thefilemodtime#1#2#3#4%
{%
    #1:#2:#3~#4%
}
```

\the@filemoddate

```
\def\the@filemoddate#1#2#3#4#5#6#7{%
    \thefilemoddate{#1}{#2}{#3}%
}
```

`\the@filemodtime`

```
\def\the@filemodtime#1#2#3{%
    \thefilemodtime
}
```

3.6 Auxiliary Macros

The 'Z' characters are changed to catcode 12 because this is how they appear in the string returned by `\pdffilemoddate`.

```
%<latex>\newcommand*\filemodZ{}
%<latex>\newcommand*\filemodz{}
\begingroup
\catcode`\D=12
```

`\filemodZ`

Holds 'Z' with catcode 12 (*other*) like it is returned by `\pdffilemoddate`. Requires use of `\csname` because 'Z' isn't a letter at the moment.

```
\expandafter\gdef\csname filemodZ\endcsname{Z}%
```

`\filemodz`

```
\let\filemodz=Z\relax

\endgroup
```

Restore catcode for plainTeX:

```
%<*tex>
\filemod@cat
\expandafter\let\csname filemod@cat\endcsname\relax
%</tex>
```