

# The filemod Package

Martin Scharrer

[martin@scharrer-online.de](mailto:martin@scharrer-online.de)

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

Version v1.2 – 2011/09/19

## 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 L<sup>A</sup>T<sub>E</sub>X compiler. It uses the `\pdffilemoddate` primitive of pdfL<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X and plain  $\epsilon$ -T<sub>E</sub>X as long pdf(L<sup>A</sup>)T<sub>E</sub>X (in PDF or DVI mode) or Lua(L<sup>A</sup>)T<sub>E</sub>X is used. X<sub>Y</sub>L<sup>A</sup>T<sub>E</sub>X is not supported because it does not provide `\pdffilemoddate`.

## Contents

<b>Contents</b>		2.5	Parsing of the file modification date . . . . .	6	
<b>1</b>	<b>Introduction</b>	<b>1</b>	2.6	Auxiliary Macros . . . . .	7
<b>2</b>	<b>Usage</b>	<b>2</b>	<b>3</b>	<b>Implementation</b>	<b>8</b>
2.1	Print File Modification Date and Time . . . . .	2	3.1	Parser . . . . .	8
2.2	Get File Modification Date and Time as Number	3	3.2	Minimal set of expandable Macros . . . . .	10
2.3	Compare File Modification Date/Time . . . . .	4	3.3	Expandable Macros . . .	12
2.4	Return Newest or Oldest File from a List . . . . .	5	3.4	Non-Expandable Macros	22
			3.5	Macros to print “today” string . . . . .	27
			3.6	Display Macros . . . . .	28
			3.7	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 L<sup>A</sup>T<sub>E</sub>X using `\usepackage{filemod}` as usual. With plain  $\epsilon$ -T<sub>E</sub>X it can be loaded using `\input filemod`. Some required internal L<sup>A</sup>T<sub>E</sub>X 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`<sup>1</sup> can be used in addition to format the dates and times in a language specific format. See also the `getfiledate`<sup>2</sup> 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): "2011/09/19 13:19:39 +02'00'"

---

<sup>1</sup>CTAN: <http://www.ctan.org/pkg/datetime>

<sup>2</sup>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: "2011/09/19"

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: "13:19:39 +02'00'"

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

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

`\Filemodtoday{<filename>}`

Prints the file modification date of the given file in the current format of `\today`. For this the compiler date is set locally to the file modification date and then `\today` is used to print that date. This takes advantages of any localised definition provided by the `babel` package or other packages.

`\FilemodToday{<filename>}`

Similar to `\FilemodToday` but also prints the full file modification date/time using the `\thefilemod` format macro. For this the `\thefilemoddate` macro is changed locally to use `\today` with the file modification date.

## 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:20110919131939+02'00'" which is the file modification date of the source file of this manual. Unfortunately this number is too 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 exist. 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 use 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:

```
1 %<*tex>
2 \expandafter\edef\csname filemod@cat\endcsname{\noexpand\catcode'\noexpand\@=\the\catcode'\@relax/
3 \catcode'\@=11
4 %</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.

```
5 \ifx\filemod@directtrue\@undefined
6 \csname newif\expandafter\endcsname\csname /
  iffilemod@direct\endcsname
7 \filemod@directtrue
8 \ifx\pdffilemoddate\@undefined
9 %<latex> \RequirePackage{pdftexcmds}
10 %<tex> \input pdftexcmds.sty
11 \filemod@directfalse
12 \ifx\pdf@filemoddate\@undefined
13 \edef\filemod@help
14 {The required command \string\pdffilemoddate\space
  space is not defined.
15 This means that the used\space\space LaTeX /
  compiler does not support it.
16 Please make sure that pdfLaTeX 1.30.0 or\space\space\space
  newer or LuaLaTeX is /
  used.
17 XeLaTeX does not support reading file /
  modification\space\space dates.
18 }%
19 %<latex> \PackageError{filemod}{Required command \string\pdffilemoddate\space
  is not defined!}{\filemod@help}
20 %<tex> \errhelp\expandafter{\filemod@help}
21 %<tex> \errmessage{filemod package: Required /
  command \string\pdffilemoddate\space is not /
  defined!}
22 \fi
23 \fi
24 \fi
```

#### 3.1 Parser



### `\filemodparse`

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

```
25 %<latex>\newcommand*\filemodparse{}
26 \iffilemod@direct
27 \def\filemodparse#1#2{%
28     \expandafter\filemod@parse\pdf@filemoddate{#2}\
        relax{#1}%
29 }
30 \else
31 \def\filemodparse#1#2{%
32     \expandafter\expandafter
33     \expandafter\filemod@parse\pdf@filemoddate{#2}\
        relax{#1}%
34 }
35 \fi
```

### `\filemod@parse`

#1: Expanded file mod date  
#2: Macro

```
36 \def\filemod@parse#1\relax#2{%
37     \ifx\relax#1\relax
38         \expandafter\@firstoftwo
39     \else
40         \expandafter\@secondoftwo
41     \fi
42     {\filemodnotexists{#2}}%
43     {\filemod@parse@#1\empty{#2}\relax}%
44 }
```

The ‘D’, ‘:’ and ‘Z’ characters are changed to catcode 12 because this is how they appear in the string returned by `\pdf@filemoddate`.

```
45 \begingroup
46 \catcode'\D=12
47 \catcode'\Z=12
48 \catcode'\:=12
```

### `\filemod@parse@`

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

```

#6: M2
#7: D1
#8: D2
#9: Rest
49 \gdef\filemod@parse@ D:#1#2#3#4#5#6#7#8#9\relax{%
50   \filemod@parse@@{\#1#2#3#4}{\#5#6}{\#7#8}}#9\relax
51 }

```

`\filemodnotexists`

```

#1: Macro provided to \filemodparse
Macro which is used for non-existing files.
52 %<latex>\newcommand*\filemodnotexists{}
53 \gdef\filemodnotexists#1{%
54   #1{0000}{00}{00}{00}{00}{00}{Z}%
55 }
56 \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.
57 \def\filemod@parse@@#1#2#3#4#5#6#7#8\empty#9\relax{%
58   #9#1{\#2#3}{\#4#5}{\#6#7}{\#8}%
59 }

```

### 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`.

```

60 \begingroup
61 \catcode ‘\D=12
62 \catcode ‘\Z=12
63 \catcode ‘\:=12

```

**\filemodNumdate**

```
64 %<latex>\newcommand*\filemodNumdate{}
65 \iffilemod@direct
66 \gdef\filemodNumdate#1{%
67     \expandafter\filemod@Numdate\pdffilemoddate{#1}D/
        :000000000000000Z\relax
68 }
69 \else
70 \gdef\filemodNumdate#1{%
71     \expandafter\expandafter
72     \expandafter\filemod@Numdate\pdf@filemoddate{#1}D/
        :000000000000000Z\relax
73 }
74 \fi
```

**\filemod@Numdate**

```
75 \gdef\filemod@Numdate D:#1#2#3#4#5#6#7#8#9\relax{%
76     #1#2#3#4#5#6#7#8%
77 }
```

**\filemodNumtime**

```
78 %<latex>\newcommand*\filemodNumtime{}
79 \iffilemod@direct
80 \gdef\filemodNumtime#1{%
81     \expandafter\filemod@Numtime\pdffilemoddate{#1}D/
        :000000000000000Z\relax
82 }
83 \else
84 \gdef\filemodNumtime#1{%
85     \expandafter\expandafter
86     \expandafter\filemod@Numtime\pdf@filemoddate{#1}D/
        :000000000000000Z\relax
87 }
88 \fi
```

**\filemod@Numtime**

```
89 \gdef\filemod@Numtime D:#1#2#3#4#5#6#7#8#9\relax{%
90     \filemod@@Numtime#9\relax
91 }
```

`\filemod@@Numtime`

```
92 \gdef\filemod@@Numtime#1#2#3#4#5#6#7\relax{%  
93   #1#2#3#4#5#6%  
94 }  
  
95 \endgroup
```

`\filemodCmp`

```
96 %<latex>\newcommand*\filemodCmp[2]%  
97 %<tex>\def\filemodCmp#1#2%  
98 {%  
99   \ifcase0%  
100     \ifnum\filemodNumdate{#2}>\filemodNumdate{#1}/  
101       1\else  
102       \ifnum\filemodNumdate{#2}=\filemodNumdate/  
103         {#1} %  
104       \ifnum\filemodNumtime{#2}>\/  
105         filemodNumtime{#1} 1\fi  
106     \fi  
107     \fi  
108     \space  
109     \expandafter\@firstoftwo  
110   \or  
111     \expandafter\@secondoftwo  
112   \fi  
113 }  
114 }
```

Some required L<sup>A</sup>T<sub>E</sub>X macros for the plainT<sub>E</sub>X version:

```
111 %<*tex>  
112 \long\def\@firstoftwo#1#2{#1}  
113 \long\def\@secondoftwo#1#2{#2}  
114 %</tex>  
  
115 %<latex>\RequirePackage{filemod-expmin}  
116 %<tex>\input filemod-expmin
```

### 3.3 Expandable Macros

#### 3.3.1 Numeric macros

`\filemodnumdate`

Simply calls the parse macro.

```

117 %<latex>\newcommand*%
118 %<tex>\def
119 \filemodnumdate{\filemodparse\filemod@numdate}

```

**\filemod@numdate**

```

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

```

```

120 % Gobbles everything except "YYYYMMDD" which is /
      returned as number without the braces.
121 \def\filemodnumdate#1#2#3#4#5#6#7{#1#2#3}

```

**\filemodnumtime**

Simply calls the parse macro.

```

122 %<latex>\newcommand*%
123 %<tex>\def
124 \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.

```

125 \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.

```
126 \def\filemod@opt#1#2#3{%
127   \expandafter
128   \remove@to@nnil@exec
129   \ifx[#3\@nnil\remove@to@nnil
130     \expandafter#1%
131   \else\@nnil\empty
132     \expandafter#2%
133     \expandafter\filemodoptdefault
134   \fi
135   {#3}%
136 }
```

`\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.

```

137 \def\remove@to@nnil@exec#1\@nnil#2{%
138     \ifx\@nnil#1\@nnil\else
139         \expandafter#2
140     \fi
141 }

```

### 3.3.3 Compare file dates

#### `\filemodcmp`

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

```

142 %<latex>\newcommand*%
143 %<tex>\def
144 \filemodcmp{%
145     \filemod@opt\filemod@cmp@opt\filemod@cmp
146 }

```

#### `\filemodoptdefault`

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

```

147 %<latex>\newcommand*%
148 %<tex>\def
149 \filemodoptdefault{1}

```

#### `\filemod@cmp@opt`

#1: '[' wrapped in {}

#2: Content of optional argument

Removes the brackets from the optional argument.

```

150 \def\filemod@cmp@opt#1#2]{%
151     \filemod@cmp{#2}%
152 }

```

#### `\filemod@cmp`

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

```

153 \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.

```

154 \def\filemod@@cmp#1#2#3#4{%
155   \ifcase0%
156     \ifnum\filemodnumdate{#4}>\filemodnumdate{#3} 1\else
157       \ifnum\filemodnumdate{#4}=\filemodnumdate{#3} %
158         \ifnum\filemodnumtime{#4}>\filemodnumtime{#3} 1\else
159           \ifnum\filemodnumtime{#4}=\filemodnumtime{#3} 2\fi
160         \fi
161       \fi
162     \fi
163   \space
164   \csname @firstoft\ifnum#2>2 hree\else wo\fi\
165     expandafter\endcsname
166   \or
167   \csname @secondoft\ifnum#2>2 hree\else wo\fi\
168     expandafter\endcsname
169   \else
170     \csname @%
171       \ifcase#2%
172         gobbletwo%
173       \or
174         firstoftwo%
175       \or
176         secondoftwo%
177       \else
178         thirdofthree%
179       \fi
180     \expandafter
181     \endcsname
182   \fi
183 }
```

`\@firstofthree`

Expands to the first of the next three arguments.



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

`\@secondofthree`

Expands to the second of the next three arguments.

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

Some required L<sup>A</sup>T<sub>E</sub>X macros for the plainT<sub>E</sub>X version:

```
184 %<*tex>
185 \long\def\@thirdofthree#1#2#3{#3}
186 \long\def\@gobble#1{}
187 \long\def\@gobbletwo#1#2{}
188 \def\remove@to@nnil#1\@nnil{}
189 %</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.

```
190 %<latex>\newcommand*%
191 %<tex>\def
192 \filemodnewest{%
193     \filemod@opt\filemod@newest@opt\filemod@newest
194 }
```

`\filemod@newest@opt`

#1: The '[' wrapped in {}

#2: Content of optional argument

Removes braces around the optional argument.

```
195 \def\filemod@newest@opt#1#2]{%
196     \filemod@newest{#2}%
197 }
```

`\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.

```

198 \def\filemod@newest#1#2#3{%
199     \filemod@@cmp>{#1}{#2}{#3}{#2}{#3}%
200 }

```

#### `\filemodoldest`

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

```

201 %<latex>\newcommand*%
202 %<tex>\def
203 \filemodoldest{%
204     \filemod@opt\filemod@oldest@opt\filemod@oldest
205 }

```

#### `\filemod@oldest@opt`

#1: The '[' wrapped in {}

#2: Content of optional argument

Removes braces around the optional argument.

```

206 \def\filemod@oldest@opt#1#2]{%
207     \filemod@oldest{#2}%
208 }

```

#### `\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.

```

209 \def\filemod@oldest#1#2#3{%
210     \filemod@@cmp<{#1}{#2}{#3}{#2}{#3}%
211 }

```

### 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.

```

212 %<latex>\newcommand*\filemodNewest{}
213 \def\filemodNewest#1#{%
214   \expandafter\expandafter
215   \expandafter\@filemodNewest
216   \csname
217     @%
218     \ifx\@nnil#1\@nnil
219       first%
220     \else
221       second%
222     \fi
223     oftwo%
224   \endcsname
225   {[\filemodoptdefault]}%
226   {#1}%
227 }

```

#### `\filemodOldest`

#1: Tokens between macros and opening brace

Like `\filemodNewest` but returns the oldest file in the given list. It and its sub-macros 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.

```

228 %<latex>\newcommand*\filemodOldest{}
229 \def\filemodOldest#1#{%
230   \expandafter\expandafter
231   \expandafter\@filemodOldest
232   \csname
233     @%
234     \ifx\@nnil#1\@nnil
235       first%
236     \else
237       second%
238     \fi
239     oftwo%
240   \endcsname
241   {[\filemodoptdefault]}%
242   {#1}%
243 }

```

#### `\@filemodNewest`

#1: Optional argument

#2: File name list

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

```

244 \def\@filemodNewest[#1]#2{%
245     \@@filemodNewest{#1}#2\filemod@end
246 }

```

#### **\@filemodOldest**

#1: Optional argument

#2: File name list

Like \@filemodNewest.

```

247 \def\@filemodOldest[#1]#2{%
248     \@@filemodOldest{#1}#2\filemod@end
249 }

```

#### **\@@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.

```

250 \def\@@filemodNewest#1#2{%
251     \filemod@Newest{#2}{#1}%
252 }

```

#### **\@@filemodOldest**

#1: Optional argument

#2: First file name

```

253 \def\@@filemodOldest#1#2{%
254     \filemod@Oldest{#2}{#1}%
255 }

```

#### **\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 \csize 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.

```

256 \def\filemod@Newest#1#2#3{%
257   \iffilemod@end{#3}%
258     {#1}%
259     {%
260       \expandafter\expandafter
261       \expandafter\expandafter
262       \expandafter\expandafter
263       \expandafter\filemod@Newest
264       \expandafter\expandafter
265       \expandafter\expandafter
266       \expandafter\expandafter
267       \expandafter{%
268         \expandafter\expandafter
269         \expandafter\@gobble
270         \expandafter\string\csname\filemod@@cmp/
271           >{#2}{#1}{#3}{#1}{#3}\endcsname}{#2}}%

```

**\filemod@Oldest**

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

Like **\filemod@Newest** but with different compare operator.

```

272 \def\filemod@Oldest#1#2#3{%
273   \iffilemod@end{#3}%
274     {#1}%
275     {%
276       \expandafter\expandafter
277       \expandafter\expandafter
278       \expandafter\expandafter
279       \expandafter\filemod@Oldest
280       \expandafter\expandafter
281       \expandafter\expandafter
282       \expandafter\expandafter
283       \expandafter{%
284         \expandafter\expandafter
285         \expandafter\@gobble
286         \expandafter\string\csname\filemod@@cmp/
287           <{#2}{#1}{#3}{#1}{#3}\endcsname}{#2}}%

```

**\iffilemod@end**

#1: Next filename or end marker  
 Checks if the argument is the **\filemod@end** marker.

```

288 \def\iffilemod@end#1{%
289   \ifx\filemod@end#1%
290     \expandafter\@firstoftwo
291   \else
292     \expandafter\@secondoftwo
293   \fi
294 }

```

`\filemod@end`

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

```

295 \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`

```

296 %<latex>\newcommand*%
297 %<tex>\def
298 \Filemodgetnum{\filemodparse\Filemod@getnum}

```

`\Filemod@getnum`

```

299 \def\Filemod@getnum#1#2#3#4#5#6#7{%
300   \def\filemoddate{#1#2#3}%
301   \def\filemodtime{#4#5#6}%
302   \def\filemodtz{#7}%
303 }

```

#### 3.4.2 Compare Two File Modification Dates

`\Filemodcmp`

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

```

304 %<*latex>
305 \newcommand\Filemodcmp[1][1]{%
306     \def\filemod@next{\filemodcmpresult}%
307     \Filemod@cmp{#1}%
308 }
309 %</latex>

```

### \FilemodCmp

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

```

310 %<*latex>
311 \newcommand\FilemodCmp[1][1]{%
312     \let\filemod@next\empty
313     \Filemod@cmp{#1}%
314 }
315 %</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.

```

316 \def\Filemod@cmp#1#2#3{%
317     \Filemodgetnum{#2}%
318     \let\filemoddatea\filemoddate
319     \let\filemodtimea\filemodtime
320     \Filemodgetnum{#3}%
321     \ifcase0%
322         \ifnum\filemoddate>\filemoddatea\space1\else
323             \ifnum\filemoddate=\filemoddatea\space
324                 \ifnum\filemodtime>\filemodtimea\space1\else
325                     \ifnum\filemodtime=\filemodtimea\space2\fi
326                 \fi
327             \fi
328         \fi
329     \relax

```

First file is newer:

```

330     \def\filemodresultfile{#1}%
331     \ifnum#1>2\relax
332         \def\filemodcmpresult##1##2##3{##1}%
333     \else
334         \let\filemodcmpresult\@firstoftwo
335     \fi
336 \or

```

Second file is newer:

```

337     \def\filemodresultfile{#2}%
338     \ifnum#1>2\relax
339         \def\filemodcmpresult##1##2##3{##2}%
340     \else
341         \let\filemodcmpresult\@secondoftwo
342     \fi
343 \else

```

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

```

344     \ifcase#1\relax
345         \let\filemodresultfile\empty
346         \let\filemodcmpresult\@gobbletwo
347     \or
348         \def\filemodresultfile{#1}%
349         \let\filemodcmpresult\@firstoftwo
350     \or
351         \def\filemodresultfile{#2}%
352         \let\filemodcmpresult\@secondoftwo
353     \else
354         \let\filemodresultfile\empty
355         \let\filemodcmpresult\@thirdofthree
356     \fi
357 \fi
358 \filemod@next
359 }

```

`\filemodcmpresult`

Defined above.

### 3.4.3 Compare file mod times and return file name

Handlers for optional arguments for plain $\text{\TeX}$ . If none is provided the `\filemodoptdefault` is used.

```

360 %<tex>
361 \def\filemod@chkopt#1{%
362     \def\filemod@optcmd{#1}%
363     \futurelet\filemod@tok\filemod@@chkopt

```



```

364 }
365 \def\filemod@@chkopt{%
366     \ifx[\filemod@tok
367         \expandafter\filemod@readopt
368     \else
369         \expandafter\filemod@optcmd
370         \expandafter\filemodoptdefault
371     \fi
372 }
373 \def\filemod@readopt[#1]{%
374     \filemod@optcmd{#1}%
375 }
376 %</tex>

```

#### **\Filemodnewest**

Simply uses **\FilemodNewest**.

```

377 %<*latex>
378 \newcommand*\Filemodnewest[3][\filemodoptdefault]{\FilemodNewest[#{1}]{#{2}{#{3}}}}
379 %</latex>
380 %<*tex>
381 \def\Filemodnewest{\filemod@chkopt\Filemod@newest}
382 \def\Filemod@newest#1#2#3{\Filemod@Newest/
383     {#1}{#{2}{#{3}}}}
384 %</tex>

```

#### **\Filemodoldest**

Simply uses **\FilemodOldest**.

```

384 %<*latex>
385 \newcommand*\Filemodoldest[3][\filemodoptdefault]{\FilemodOldest[#{1}]{#{2}{#{3}}}}
386 %</latex>
387 %<*tex>
388 \def\Filemodoldest{\filemod@chkopt\Filemod@oldest}
389 \def\Filemod@oldest#1#2#3{\Filemod@Oldest/
390     {#1}{#{2}{#{3}}}}
391 %</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.

```

391 %<latex>\newcommand*\FilemodNewest[2][\filemodoptdefault]%
392 %<tex>\def\FilemodNewest{\filemod@chkopt\
Filemod@Newest}
393 %<tex>\def\Filemod@Newest#1#2%
394 {%
395     \def\filemode@tie{#1}%
396     \def\filemod@gl{>}%
397     \Filemod@est#2\filemod@end
398 }

```

#### `\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.

```

399 %<latex>\newcommand*\FilemodOldest[2][\filemodoptdefault]%
400 %<tex>\def\FilemodOldest{\filemod@chkopt\
Filemod@Oldest}
401 %<tex>\def\Filemod@Oldest#1#2%
402 {%
403     \def\filemode@tie{#1}%
404     \def\filemod@gl{<}%
405     \Filemod@est#2\filemod@end
406 }

```

#### `\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.

```

407 \def\Filemod@est#1{%
408     \def\filemodresultfile{#1}%
409     \Filemodgetnum{#1}%
410     \let\filemodresultdate\filemoddate
411     \let\filemodresulttime\filemodtime
412     \Filemod@@est
413 }

```

#### `\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.

```

414 \def\Filemod@@est#1{%
415   \iffilemod@end{#1}{}{%
416     \Filemodgetnum{#1}%
417     \ifcase0%
418       \ifnum\filemoddate\filemod@gl\
419         filemodresultdate\space1\else
420         \ifnum\filemoddate=\filemodresultdate\
421           space
422           \ifnum\filemodtime\filemod@gl\
423             filemodresulttime\space1\else
424             \ifnum\filemodtime=\
425               filemodresulttime\space
426               \ifnum\filemode@tie=1\else 1\
427               fi
428             \fi
429           \fi
430         \fi
431       \else
432         \def\filemodresultfile{#1}%
433         \let\filemodresultdate\filemoddate
434         \let\filemodresulttime\filemodtime
435       \fi
436     \Filemod@@est
437   }%
438 }

```

`\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.

```

435 \def\filemod@gl{>}

```

### 3.5 Macros to print “today” string

`\Filemodtoday`

```

436 %<latex>\newcommand*\Filemodtoday[1]%
437 %<tex>\def\Filemodtoday#1%
438 {%
439   \def\thefilemod##1##2##3##4##5##6##7{\year##1 \
440     month##2 \day##3 \today}%
441   \filemodprint{#1}%
442 }%

```

**\FilemodToday**

```
442 %<latex>\newcommand*\FilemodToday[1]%
443 %<tex>\def\FilemodToday#1%
444 {%
445     \def\thefilemoddate##1##2##3{\year##1 \month##2 \/
        day##3 \today}%
446     \filemodprint{#1}%
447 }}
```

### 3.6 Display Macros

**\filemodprint**

```
448 %<latex>\newcommand*
449 %<tex>\def
450 \filemodprint{\filemodparse\thefilemod}
```

**\filemodprintdate**

```
451 %<latex>\newcommand*
452 %<tex>\def
453 \filemodprintdate{\filemodparse\the@filemoddate}
```

**\filemodprinttime**

```
454 %<latex>\newcommand*
455 %<tex>\def
456 \filemodprinttime{\filemodparse\the@filemodtime}
```

**\thefilemod**

```
457 %<latex>\newcommand*\thefilemod[7]%
458 %<tex>\def\thefilemod#1#2#3#4#5#6#7%
459 {%
460     \thefilemoddate{#1}{#2}{#3}%
461     \filemodsep
462     \thefilemodtime{#4}{#5}{#6}{#7}%
463 }

464 %<latex>\newcommand*\filemodsep{ }
465 %<tex>\let\filemodsep\space
```

`\thefilemoddate`

```
466 %<latex>\newcommand*\thefilemoddate[3]%
467 %<tex>\def\thefilemoddate#1#2#3%
468 {#1/#2/#3}
```

`\thefilemodtime`

```
469 %<latex>\newcommand*\thefilemodtime[4]%
470 %<tex>\def\thefilemodtime#1#2#3#4%
471 {%
472     #1:#2:#3~#4%
473 }
```

`\the@filemoddate`

```
474 \def\the@filemoddate#1#2#3#4#5#6#7{%
475     \thefilemoddate{#1}{#2}{#3}%
476 }
```

`\the@filemodtime`

```
477 \def\the@filemodtime#1#2#3{%
478     \thefilemodtime
479 }
```

### 3.7 Auxiliary Macros

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

```
480 %<latex>\newcommand*\filemodZ{}
481 %<latex>\newcommand*\filemodz{}
482 \begingroup
483 \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.

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

`\filemodz`

```
485 \let\filemodz=Z\relax
```

```
486 \endgroup
```

Restore catcode for plainTeX:

```
487 %<*tex>
```

```
488 \filemod@cat
```

```
489 \expandafter\let\csname filemod@cat\endcsname\relax
```

```
490 %</tex>
```