The *filehook* Package

Martin Scharrer

martin@scharrer-online.de

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

Version v0.3 – 2010/12/20

Abstract

This small package provides hooks for input files. Document and package authors can use these hooks to execute code at begin or the end of specific or all input files.

1 Introduction

These package changes some internal LaTeX macros used to load input files so that they include 'hooks'. A hook is an (internal) macro executed at specific points. Normally it is initially empty, but can be extended using an user level macro. The most common hook in LaTeX is the 'At-Begin-Document' hook. Code can be added to this hook using `\AtBeginDocument{`⟨*TEXcode*⟩`}`.

2 Usage

This package provides several groups of hooks: for file read using `\input`, for files read using `\include` and for all read files (i.e. all files read using `\InputIfFileExists`, which includes package and class files and files falling into the first two groups). Since v0.3 from 2010/12/20 there are also hooks for package and class files. All groups include a 'AtBegin' and a 'AtEnd' macro. The `\include` group has also a 'After' hook which it is executed *after* the page break (`\clearpage`) is inserted by the `\include` code. In contrast, the 'AtEnd' hook is executed before the trailing page break and the 'AtBegin' hook is executed after the *leading* page break.

The first three groups includes general and file specific hooks. The general hooks are executed for every file of this group and provide the file name as argument `#1`. The file specific ones are only executed for a certain file. The package and class hocks are always specific to one file.

The below macros can be used to add material (TeX code) to the related hooks. All 'AtBegin' macros will *append* the code to the hooks, but the 'AtEnd' and 'After' macros will *prefix* the code instead. This ensures that two different packages adding material in 'AtBegin'/'AtEnd' pairs do not overlap each other. Instead the later used package adds the code closer to the file content,

'inside' the material added by the first package. Therefore it is safely possible to surround the content of a file with multiple LaTeX environments using multiple 'AtBegin'/'AtEnd' macro calls. If required inside another package a different order can be enforced by using the internal hook macros shown in the implementation section.

Include Files

```
\AtBeginOfIncludes{⟨TEX code⟩}
\AtEndOfIncludes{⟨TEX code⟩}
\AfterIncludes{⟨TEX code⟩}
```

All these macro take one argument (some TeX code) which is added to the specific hook for files read using `\include`. The code can use the macro argument `#1` which will be expanded to the include ⟨*file name*⟩, i.e. the hooks are macros with one argument which will be the file name. As described above the 'AtEnd' hook is executed before and the 'After' hook is executed after the trailing `\clearpage`. Material which should be (still) valid in the page header or footer of the last page of such an file should therefore use the 'After' hook.

```
\AtBeginOfIncludeFile{⟨file name⟩}{⟨TEX code⟩}
\AtEndOfIncludeFile{⟨file name⟩}{⟨TEX code⟩}
\AfterIncludeFile{⟨file name⟩}{⟨TEX code⟩}
```

These file-specific macros take the two arguments. The ⟨*code*⟩ is only executed for the file with the given ⟨*file name*⟩ and only if it is read using `\include`. It is not allowed to use macro arguments inside the code. The ⟨*file name*⟩ should be identical to the name used for `\include` and not include the '`.tex`' extension.

Input Files

```
\AtBeginOfInputs{⟨TEX code⟩}
\AtEndOfInputs{⟨TEX code⟩}
```

Like the `\...OfIncludes{`⟨*code*⟩`}` macros above, just for file read using `\input`. Again, the macro argument `#1` can be used inside the ⟨*code*⟩ and will be expanded to the ¡file name¿.

```
\AtBeginOfInputFile{⟨file name⟩}{⟨TEX code⟩}
\AtEndOfInputFile{⟨file name⟩}{⟨TEX code⟩}
```

Like the `\...OfIncludeFile{`⟨*¡file name*⟩¿`}{`⟨*code*⟩`}` macros above, just for file read using `\input`. Here the ⟨*file name*⟩ should include the file extension! The ⟨*code*⟩ must not include any macro arguments (`#1`).

All Files

```
\AtBeginOfFiles{⟨TEX code⟩}
\AtEndOfFiles{⟨TEX code⟩}
```

These macros add the given {⟨*code*⟩} to two hooks executed for all files read using the `\InputIfFileExists` macro. This macro is used internally by the `\input`, `\include` and `\usepackage`/`\RequirePackage` macros. Packages and classes might use it to include additional or auxiliary files. Authors can exclude those files from the hooks by using `\IfFileExists{`⟨*file name*⟩`}{\@input\@filef@und}{}` instead.

```
\AtBeginOfFile{⟨file name with extension⟩}{⟨TEX code⟩}
\AtEndOfFile{⟨file name with extension⟩}{⟨TEX code⟩}
```

Like the `\...OfIncludeFile{`⟨*file name*⟩`}{`⟨*TEX code*⟩`}` macros above, just for 'all' read files. Here the ⟨*file name*⟩ should include the file extension! The ⟨*code*⟩ must not include any macro arguments (`#1`).

The 'all files' hooks are closer to the file content than the `\input` and `\include` hook, i.e. the `\AtBeginOfFiles` comes *after* the `\AtBeginOfIncludes` and the `\AtEndOfFiles` comes *before* the `\AtEndOfIncludes` hook.

Package Files

```
\AtBeginOfPackageFile{⟨package name⟩}{⟨TEX code⟩}
\AtEndOfPackageFile*{⟨package name⟩}{⟨TEX code⟩}
```

This macros install the given ⟨*TEX code*⟩ in the 'AtBegin' and 'AtEnd' hooks of the given package file. The `\AtBeginOfPackageFile` simply executes `\AtBeginOfFile{`⟨*package name*⟩`.sty}{`⟨*TEXcode*⟩`}`. Special care is taken to ensure that the 'AtEnd' code is executed *after* any code installed by the package itself using the LaTeX macro `\AtEndOfPackage`. If the starred version is used and the package is already loaded the code is executed right away.

Class Files

```
\AtBeginOfClassFile{⟨class name⟩}{⟨TEX code⟩}
\AtEndOfClassFile*{⟨class name⟩}{⟨TEX code⟩}
```

This macros install the given ⟨*TEX code*⟩ in the 'AtBegin' and 'AtEnd' hooks of the given class file. They work with classes loaded using `\LoadClass`, `\LoadClassWithOptions` and also `\documentclass`. However, in the latter case `filehook` must be loaded using `\RequirePackage` beforehand. The `\AtBeginOfClassFile`

simply executes \AtBeginOfFile{⟨*class name*⟩.cls}{⟨*TEXcode*⟩}. Special care is taken to ensure that the 'AtEnd' code is executed *after* any code installed by the class itself using the LaTeX macro \AtEndOfPackage. If the starred version is used and the class is already loaded the code is executed right away.

3 Compatibility Issues with other Packages

The filehook package might clash with other packages or classes which also redefine \InputIfFileExists. Special compatibility code is in place for the known packages listed below (in their current implementation). If any other unknown definition is found an error will be raised. The package option 'force' can be used to prevent this and to force the redefinition of this macro. Then any previous modifications will be lost, which will most likely break the other package. Please do not hesitate to inform the author of filehook of any encountered problems with other packages.

jmlrbook

The jmlrbook class from the jmlr bundle temporary redefines \InputIfFileExists to import papers. The 'original' definition is saved away at load time of the package and is used internally by the new definition. This means that the hooks will not be active for this imported files because filehook is loaded after the class. This should not affect its normal usage. Note that, in theory, the package could be loaded before \documentclass using \RequirePackage to enable the file hooks also for these files.

memoir

The memoir class redefines \InputIfFileExists to add own hooks identical to the At...OfFiles hooks (there called \AtBeginFile and \AtEndFile). This hooks will be moved to the corresponding ones of filehook and will keep working as normal.

scrlfile

The scrlfile package from the *koma-script* bundle redefines \InputIfFileExists to allow file name aliases and to also add hooks. If required it should be loaded before filehook, which will add its hooks correctly to the modified definition.

fink

The filehook and currfile packages where written as replacements for the fink package, where filehook provides the necessary hooks for currfile. The fink package has now been deprecated in favour of currfile and should not be used anymore. The fink compatibility code has been removed from

filehook and both cannot be used successfully together as both redefine the \InputIfFileExists macro.

listings

The listings package uses \input inside \lstinputlisting. Therefore the InputFile(s) and File(s) hooks are also triggered for these files. Please note that this hooks are executing inside a verbatim environment. While the code in the hook is not affected (because it was added outside the verbatim environment), any further code read using any input macro (\input, \@input, \@@input(TEX's \input{)}, ...) will be processed verbatim and typeset as part of the listing. A known package suffering from this is svn-multi which loads .svx files for every .tex file. A workaround for this issue is to temporally redefine \input to \@input for \lstinputlisting: {\let\input\@input\lstinputlisting{...}}.

4 Implementation

4.1 Options

```
\newif\iffilehook@force
\DeclareOption{force}{\filehook@forcetrue}
\ProcessOptions\relax
```

4.2 Installation of Hooks

The \@input@ and \@iinput macros from latex.ltx are redefined to install the hooks.

First the original definitions are saved away.

```
\let\filehook@orig@@input@\@input@
\let\filehook@orig@@iinput\@iinput
```

\@input@

This macro is redefined for the \include file hooks. Checks if the next command is \clearpage which indicates that we are inside \@include. If so the hooks are installed, otherwise the original macro is used unchanged. For the 'after' hook an own \clearpage is inserted and the original one is gobbled.

```
\def\@input@#1{%
  \@ifnextchar\clearpage
    {\filehook@include@atbegin{#1}%
     \filehook@orig@@input@{#1}%
     \filehook@include@atend{#1}%
     \clearpage
```

```
      \filehook@include@after{#1}%
      \@gobble
    }%
    {\filehook@orig@@input@{#1}}%
}
```

\@iinput

This macro is redefined for the \input file hooks. it simply surrounds the original macro with the hooks.

```
\def\@iinput#1{%
  \filehook@input@atbegin{#1}%
  \filehook@orig@@iinput{#1}%
  \filehook@input@atend{#1}%
}
```

\InputIfFileExists

This macro is redefined for the general file hooks. The original definition is checked but is not saved away and called by the new definition, because of the existing complexity. The hooks must be places around the actual input macro (\@@input).

Alternatives definitions of \InputIfFileExists are defined here for comparison. This is done inside a group to keep them only temporary.

```
\begingroup

\long\def\latex@InputIfFileExists#1#2{%
  \IfFileExists{#1}%
    {#2\@addtofilelist{#1}%
     \@@input\@filef@und
    }%
}
\long\def\memoir@InputIfFileExists#1#2{%
  \IfFileExists{#1}%
    {#2\@addtofilelist{#1}\m@matbeginf{#1}%
     \@@input \@filef@und
     \m@matendf{#1}%
     \killm@matf{#1}}%
}
\long\def\scrlfile@InputIfFileExists#1#2{%
  \begingroup\expandafter\expandafter\expandafter\↙
     endgroup
  \expandafter\ifx\csname #1-@alias\endcsname\relax
```

```
    \expandafter\@secondoftwo
  \else
    \scr@replacefile@msg{\csname #1-@alias\endcsname}{#1}%
    \expandafter\@firstoftwo
  \fi
  {%
    \expandafter\InputIfFileExists\expandafter{\csname
      #1-@alias\endcsname}{#2}%
  }%
  {\IfFileExists{#1}{%
      \scr@load@hook{before}{#1}%
      #2\@addtofilelist{#1}%
      \@@input \@filef@und
      \scr@load@hook{after}{#1}%
    }}%
}
```

If the `scrlfile` package definition is detected the `filehook`s are added to that definition. Unfortunately the `\scr@load@hook{before}` hook is placed *before* not after the `#2\@addtofilelist{#1}` code. Otherwise the `filehook`s could simply be added to these hooks. Note that should `scrlfile` ever change its `\InputIfFileExists` macro this code will not be executed and the general clause below will kick in.

```
\ifx\InputIfFileExists\scrlfile@InputIfFileExists

\long\gdef\InputIfFileExists#1#2{%
  \begingroup\expandafter\expandafter\expandafter\endgroup
  \expandafter\ifx\csname #1-@alias\endcsname\relax
    \expandafter\@secondoftwo
  \else
    \scr@replacefile@msg{\csname #1-@alias\endcsname}{#1}%
    \expandafter\@firstoftwo
  \fi
  {%
    \expandafter\InputIfFileExists\expandafter{\csname
      #1-@alias\endcsname}{#2}%
  }%
  {\IfFileExists{#1}{%
      \scr@load@hook{before}{#1}%
      #2\@addtofilelist{#1}%
      \filehook@atbegin{#1}%
```

```
      \@@input \@filef@und
      \filehook@atend{#1}%
      \scr@load@hook{after}{#1}%
    }}%
}

\PackageInfo{filehook}{Package 'scrlfile' detected ↙
   and compensated for.}
```

Otherwise the normal `filehook` definition will be set. If `memoir` is detected its hooks are added to the appropriate `At...OfFiles` hooks. This works fine because its hooks have the exact same position.

```
\else

\ifx\InputIfFileExists\memoir@InputIfFileExists
  \AtEndOfPackage{%
    \AtBeginOfFiles{\m@matbeginf{#1}}%
    \AtEndOfFiles{\m@matendf{#1}\killm@matf{#1}}%
  }
  \PackageInfo{filehook}{Detected 'memoir' class: the↙
     memoir hooks will be moved to the 'At...OfFiles↙
    ' hooks.}
\else
```

Finally, if no specific alternate definition is detected the original LaTeX definition is checked for and a warning is given if any other unknown definition is detected. In this case it will be simply overwritten.

```
\ifx\InputIfFileExists\latex@InputIfFileExists
\else
  \@ifpackageloaded{scrlfile}{%
    \PackageWarning{filehook}{Detected 'scrlfile' ↙
       package with unknown definition of \string\↙
       InputIfFileExists}%
  }{}%

  \@ifclassloaded{memoir}{%
    \PackageWarning{filehook}{Detected 'memoir' class↙
        with unknown definition of \string\↙
       InputIfFileExists}%
  }{}%

  \iffilehook@force
    \PackageWarning{filehook}
        {Changed definition of \string\↙
           InputIfFileExists\space detected!^^J%
```

```
        The 'force' option is in effect and therefore↙
            this macros is redefined.
        This might break other packages or code}%
  \else
    \PackageError{filehook}
        {Changed definition of \string\↙
           InputIfFileExists\space detected!^^J%
        Use the 'force' option to force the ↙
           redefinition of this macro.^^J%
        This might break other packages or code}%
  \fi
\fi
\fi

\long\gdef\InputIfFileExists#1#2{%
  \IfFileExists{#1}%
    {#2\@addtofilelist{#1}%
     \filehook@atbegin{#1}%
     \@@input\@filef@und
     \filehook@atend{#1}%
    }%
}

\fi

\endgroup
```

4.3 Initialisation of Hooks

The general hooks are initialised to call the file specific hooks.

`\filehook@include@atbegin`

`\filehook@include@atend`

`\filehook@include@after`

```
\def\filehook@include@atbegin#1{%
  \@nameuse{\filehook@include@atbegin@#1}%
}
\def\filehook@include@atend#1{%
```

```
  \@nameuse{\filehook@include@atend@#1}%
}
\def\filehook@include@after#1{%
  \@nameuse{\filehook@include@after@#1}%
}
```

`\filehook@input@atbegin`

`\filehook@input@atend`

```
\def\filehook@input@atbegin#1{%
  \@nameuse{\filehook@input@atbegin@#1}%
}
\def\filehook@input@atend#1{%
  \@nameuse{\filehook@input@atend@#1}%
}
```

`\filehook@atbegin`

`\filehook@atend`

```
\def\filehook@atbegin#1{%
  \@nameuse{\filehook@atbegin@#1}%
}
\def\filehook@atend#1{%
  \@nameuse{\filehook@atend@#1}%
}
```

4.4 Hook Modification Macros

The following macros are used to modify the hooks, i.e. to prefix or append code to them.

Internal Macros

The macro prefixes for the file specific hooks are stored in macros to reduce the number of tokens in the following macro definitions.

```
\def\filehook@include@atbegin@{↙
   filehook@include@atbegin@}
\def\filehook@include@atend@{filehook@include@atend@}
\def\filehook@include@after@{filehook@include@after@}
\def\filehook@input@atbegin@{filehook@input@atbegin@}
\def\filehook@input@atend@{filehook@input@atend@}
\def\filehook@input@after@{filehook@input@after@}
\def\filehook@atbegin@{filehook@atbegin@}
\def\filehook@atend@{filehook@atend@}
\def\filehook@after@{filehook@after@}
```

\filehook@append

Uses default LaTeX macro.

```
\def\filehook@append{\g@addto@macro}
```

\filehook@appendwarg

Appends code with one macro argument. The \@tempa intermediate step is required because of the included ##1 which wouldn't correctly expand otherwise.

```
\long\def\filehook@appendwarg#1#2{%
  \begingroup
    \toks@\expandafter{#1{##1}#2}%
    \edef\@tempa{\the\toks@}%
    \expandafter\gdef\expandafter#1\expandafter##\↙
       expandafter1\expandafter{\@tempa}%
  \endgroup
}
```

\filehook@prefix

Prefixes code without an argument to a hook.

```
\long\def\filehook@prefix#1#2{%
  \begingroup
    \@temptokena{#2}%
    \toks@\expandafter{#1}%
    \xdef#1{\the\@temptokena\the\toks@}%
  \endgroup
}
```

`\filehook@prefixwarg`

Prefixes code with an argument to a hook.

```
\long\def\filehook@prefixwarg#1#2{%
  \begingroup
    \@temptokena{#2}%
    \toks@\expandafter{#1{##1}}%
    \edef\@tempa{\the\@temptokena\the\toks@}%
    \expandafter\gdef\expandafter#1\expandafter##\↙
        expandafter1\expandafter{\@tempa}%
  \endgroup
}
```

User Level Macros

The user level macros simple use the above defined macros on the appropriate hook.

`\AtBeginOfIncludes`

```
\newcommand*\AtBeginOfIncludes{%
  \filehook@appendwarg\filehook@include@atbegin
}
```

`\AtEndOfIncludes`

```
\newcommand*\AtEndOfIncludes{%
  \filehook@prefixwarg\filehook@include@atend
}
```

`\AfterOfIncludes`

```
\newcommand*\AfterIncludes{%
  \filehook@prefixwarg\filehook@include@after
}
```

`\AtBeginOfIncludeFile`

```
\newcommand*\AtBeginOfIncludeFile[1]{%
  \@ifundefined{\filehook@include@atbegin@#1.tex}%
    {\long\global\@namedef{\filehook@include@atbegin@#1.tex}}%
    {\expandafter\filehook@append\csname\filehook@include@atbegin@#1.tex\endcsname}%
}
```

\AtEndOfIncludeFile

```
\newcommand*\AtEndOfIncludeFile[1]{%
  \@ifundefined{\filehook@include@atend@#1.tex}%
    {\long\global\@namedef{\filehook@include@atend@#1.tex}}%
    {\expandafter\filehook@prefix\csname\filehook@include@atend@#1.tex\endcsname}%
}
```

\AfterOfIncludeFile

```
\newcommand*\AfterOfIncludeFile[1]{%
  \@ifundefined{\filehook@include@after@#1.tex}%
    {\long\global\@namedef{\filehook@include@after@#1.tex}}%
    {\expandafter\filehook@prefix\csname\filehook@include@after@#1.tex\endcsname}%
}
```

\AtBeginOfInputs

```
\newcommand*\AtBeginOfInputs{%
  \filehook@appendwarg\filehook@input@atbegin
}
```

\AtEndOfInputs

```
\newcommand*\AtEndOfInputs{%
  \filehook@prefixwarg\filehook@input@atend
}
```

\AtBeginOfInputFile

```
\newcommand*\AtBeginOfInputFile[1]{%
  \@ifundefined{\filehook@input@atbegin@#1}%
    {\long\global\@namedef{\filehook@input@atbegin@#1}}%
    {\expandafter\filehook@append\csname\filehook@input@atbegin@#1\endcsname}%
}
```

\AtEndOfInputFile

```
\newcommand*\AtEndOfInputFile[1]{%
  \@ifundefined{\filehook@input@atend@#1}%
    {\long\global\@namedef{\filehook@input@atend@#1}}%
    {\expandafter\filehook@prefix\csname\filehook@input@atend@#1\endcsname}%
}
```

\AtBeginOfFiles

```
\newcommand*\AtBeginOfFiles{%
  \filehook@appendwarg\filehook@atbegin
}
```

\AtEndOfFiles

```
\newcommand*\AtEndOfFiles{%
  \filehook@prefixwarg\filehook@atend
}
```

\AtBeginOfFile

```
\newcommand*\AtBeginOfFile[1]{%
  \@ifundefined{\filehook@atbegin@#1}%
    {\long\global\@namedef{\filehook@atbegin@#1}}%
    {\expandafter\filehook@append\csname\filehook@atbegin@#1\endcsname}%
}
```

\AtEndOfFile

```
\newcommand*\AtEndOfFile[1]{%
  \@ifundefined{\filehook@atend@#1}%
    {\long\global\@namedef{\filehook@atend@#1}}%
    {\expandafter\filehook@prefix\csname\↙
       filehook@atend@#1\endcsname}%
}
```

\AtBeginOfPackageFile

#1: package name

Simply add the package extension and calls the general macro.

```
\newcommand*\AtBeginOfPackageFile[1]{%
    \AtBeginOfFile{#1.\@pkgextension}%
}
```

\AtEndOfPackageFile

```
\newcommand*\AtEndOfPackageFile{%
    \@ifnextchar*\AtEndOfPackageFile@star\↙
       AtEndOfPackageFile@normal
}
```

\AtEndOfPackageFile@star

#1: package name

#2: code

If the package is already loaded the code is executed right away. Otherwise it is installed normally.

```
\def\AtEndOfPackageFile@star*#1#2{%
    \@ifpackageloaded{#1}%
        {#2}%
        {\AtEndOfPackageFile@normal{#1}{#2}}%
}
```

\AtEndOfPackageFile@normal

#1: package name

#2: code

Installs the code at the end of the package file inside a \AtEndOfPackage command to ensure it is executed after any \AtEndOfPackage code installed by the package itself.

Note if the package was already loaded or is not loaded at all the installed code is never executed.

```
\def\AtEndOfPackageFile@normal#1#2{%
    \AtEndOfFile{#1.\@pkgextension}{\AtEndOfPackage
        {#2}}%
}
```

\AtBeginOfClassFile

#1: class name

Simply add the class extension and calls the general macro.

```
\newcommand*\AtBeginOfClassFile[1]{%
    \AtBeginOfFile{#1.\@clsextension}%
}
```

\AtEndOfClassFile

#1: class name
#2: code

```
\newcommand*\AtEndOfClassFile{%
    \@ifnextchar*\AtEndOfClassFile@star\
        AtEndOfClassFile@normal
}
```

\AtEndOfClassFile@star

#1: class name
#2: code

If the class is already loaded the code is executed right away. Otherwise it is installed normally.

```
\def\AtEndOfClassFile@star*#1#2{%
    \@ifclassloaded{#1}%
        {#2}%
        {\AtEndOfClassFile@normal{#1}{#2}}%
}
```

\AtEndOfClassFile@normal

#1: class name
#2: code

Installs the code at the end of the class file inside a \AtEndOfClass command to ensure it is executed after any \AtEndOfClass code installed by the class itself.

Note if the class was already loaded or is not loaded at all the installed code is never executed.

```
298 \def\AtEndOfClassFile@normal#1#2{%
299     \AtEndOfFile{#1.\@clsextension}{\AtEndOfClass↙
        {#2}}%
300 }
```