A Document Class and a Package for Handling Multi-File Projects

Federico Garcia, Gernot Salzer

2020/10/03 v2.0

Abstract

The `subfiles` package allows authors to split a document into one main file and several subsidiary files (subfiles) akin to the `\input` command, with the added benefit of making the subfiles compilable on their own. This is achieved by reusing the preamble of the main file also for the subfiles.

Contents

1 Introduction **2**

2 Basic usage **2**

3 Advanced usage **3**
- 3.1 Including files instead of inputting them 3
- 3.2 Fixing pathes 3
- 3.3 Conditional execution of commands 4
- 3.4 Unusual locations for placing definitions and text 4

4 Use cases **5**
- 4.1 Hierarchy of directories 5
- 4.2 Cross-referencing between subfiles 6
- 4.3 Avoiding extra spaces 6

5 Troubleshooting **6**

6 Dependencies **7**

7 Version history **7**

8 The Implementation **8**
- 8.1 The class 8
- 8.2 The package 9

1 Introduction

The LaTeX commands `\include` and `\input` allow the user to split the TeX source of a document into several input files. This is useful when creating documents with many chapters, but also for handling large tables, figures and code samples, which require a considerable amount of trial-and-errors.

In this process the rest of the document is of little use, and can even interfere. For example, error messages may indicate not only the wrong line number, but may point to the wrong file. Frequently, one ends up wanting to work only on the new file:

- Create a new file, and copy-paste the preamble of the main file into it.
- Work on this file, typeset it *alone* as many times as necessary.
- Finally, when the result is satisfactory, delete the preamble from the file (alongside with `\end{document}`!), and `\include` or `\input` it from the main file.

It is desirable to reduce these three steps to the interesting, middle one. Each new, subordinate file (henceforth 'subfile') should behave both as a self-sufficient LaTeX document and as part of the whole project, depending on whether it is LaTeXed individually or `\included`/`\input` from the main document. This is what the class `subfiles.cls` and the package `subfiles.sty` are intended for.

2 Basic usage

`subfiles.sty` The main file, i.e., the file with the preamble to be shared with the subfiles, has to load the package `subfiles`:

```
\documentclass[...]{...}
\usepackage{subfiles}
\begin{document}
```

`\subfile` Subordinate files (subfiles) are loaded from the main file or from other subfiles with the command

`\subfile{`⟨*subfile_name*⟩`}`

`subfiles.cls` The subfiles have to start with the line

`\documentclass[`⟨*main_file_name*⟩`]{subfiles}`

which loads the class `subfiles`. Its only 'option', which is actually mandatory, gives the name of the main file. This name follows TeX conventions: `.tex` is the default extension, the path has to be provided if the main file is in a different directory, and directories in the path have to be separated by `/` (not `\`). Thus, we have the following structure.

main file
`\documentclass[...]{...}`
⟨*shared preamble*⟩
`\usepackage{subfiles}`
`\begin{document}`
...
`\subfile{`⟨*subfile_name*⟩`}`
...
`\end{document}`

subfile
`\documentclass[`⟨*main_file_name*⟩`]{subfiles}`
`\begin{document}`
...
`\end{document}`

Now there are two possibilities.

- If LaTeX is run on the subfile, the line `\documentclass[..]{subfiles}` is replaced by the preamble of the main file (including its `\documentclass` command). The rest of the subfile is processed normally.
- If LaTeX is run on the main file, the subfile is loaded like with an `\input` command, except that the preamble of the subfile up to `\begin{document}` as well as `\end{document}` and the lines following it are ignored.

3 Advanced usage

3.1 Including files instead of inputting them

`\subfileinclude` In plain LaTeX, you can use either `\input` or `\include` to load a file. In most cases the first is appropriate, but sometimes there are reasons to prefer the latter. Internally, the `\subfile` command uses `\input`. For those cases where you need `\include`, the package provides the command

`\subfileinclude{`⟨*subfile_name*⟩`}`

3.2 Fixing pathes

`\subfix` Whenever an error message of LaTeX or an external program indicates that a file cannot be found, the reason may be that the missing file has to be addressed by varying pathes, depending on which file is typeset. In such a case, it may help to apply the command `\subfix` to the file or path names. Examples:

package	command when used with `subfiles`
`biblatex`	`\addbibresource{\subfix{`⟨*file*⟩`}}`
`bibunits`	`\putbib[\subfix{`⟨*file1*⟩`},\subfix{`⟨*file2*⟩`},...]`
	`\defaultbibliography{\subfix{`⟨*file1*⟩`},...}`

`\bibliography` `\graphicspath` Some commands already apply the fix on the fly. At the moment these are the standard LaTeX command `\bibliography` and `\graphicspath` from the `graphics`/`graphicx` package.

3.3 Conditional execution of commands

\ifSubfilesClassLoaded The command \ifSubfilesClassLoaded is useful to execute commands conditionally, depending on whether the main file is typeset or a subfile.

```
\ifSubfilesClassLoaded{% then branch
  ... commands executed when the subfile is typeset ...
}{% else branch
  ... commands executed when the main file is typeset ...
}
```

As an example, this can be used to add the bibliography to the main document or to the subdocument, whichever is typeset:

main file	subfile
\documentclass[...]{...}	\documentclass[⟨*main_file_name*⟩]{subfiles}
\usepackage{subfiles}	\begin{document}
\bibliographystyle{alpha}	...
\begin{document}	\ifSubfilesClassLoaded{%
...	\bibliography{bibfile}%
\subfile{⟨*subfile_name*⟩}	}{}
...	\end{document}
\bibliography{bibfile}	
\end{document}	

3.4 Unusual locations for placing definitions and text

Starting with version 2.0, the subfiles package treats sub-preambles and text after \end{document} as one would expect: The preamble of subfiles is skipped when loaded with \subfile, and everything after \end{document} is ignored. In most cases this is what you want.

[v1] For reasons of compatibility, the option v1 restores the behaviour of older versions:

```
\usepackage[v1]{subfiles}
```

This will have three effects.

Code after the end of the main document is added to the preamble of the subfiles, but is ignored when typesetting the main file. Here, one can add commands that are to be processed as part of the preamble when the subfiles are typeset on their one. But this also means that any syntax error after \end{document} will ruin the LaTeXing of the subfile(s).

Code in the preamble of a subfile is processed as part of the text when typesetting the main file, but as part of the preamble when typesetting the subfile. This means that with the option v1, the preamble of a subfile can only contain stuff that is acceptable for both, the preamble and the text area. One should also keep in mind that each subfile is input within a group, so definitions made here may not work outside.

Code after \end{document} in a subfile is treated like the code preceding it when the subfile is loaded from the main file, but is ignored when typesetting the subfile. The code after `\end{document}` behaves as if following the `\subfile` command in the main file, except that it is still part of the group enclosing the subfile. As a consequence, empty lines at the end of the subfile lead to a new paragraph in the main document, even if the `\subfile` command is immediately followed by text.

4 Use cases

4.1 Hierarchy of directories

Sometimes it is desirable to put a subfile together with its images and supplementary files into its own directory. The difficulty now is that these additional files have to be addressed by different pathes depending on whether the main file or the subfile is typeset. As of version 1.3, the `subfiles` package handles this problem by using the `import` package.

As an example, consider the following hierarchy of files:

```
main.tex
mypreamble.tex
dir1/subfile1.tex
dir1/image1.jpg
dir1/text1.tex
dir1/dir2/subfile2.tex
dir1/dir2/image2.jpg
dir1/dir2/text2.tex
```

where `main`, `subfile1`, and `subfile2` have the following contents:

main.tex

```
\documentclass{article}
\input{mypreamble}
\usepackage{graphicx}
\usepackage{subfiles}
\begin{document}
\subfile{dir1/subfile1}
\end{document}
```

subfile1.tex

```
\documentclass[../main]{subfiles}
\begin{document}
\input{text1}
\includegraphics{image1.jpg}
\subfile{dir2/subfile2}
\end{document}
```

subfile2.tex

```
\documentclass[../../main]{subfiles}
\begin{document}
\input{text2}
\includegraphics{image2.jpg}
\end{document}
```

Then each of the three files can be typeset individually in its respective directory, where LaTeX is able to locate all included text files and images.

4.2 Cross-referencing between subfiles

When working with multiple subfiles under a main file, say `main.tex`, one may want to refer in subfile `A.tex` to labels in subfile `B.tex`. To make this work, load the package `xr` in the preamble of the main file and add an `\externaldocument` command after loading the `subfiles` package:

```
\usepackage{xr}
\usepackage{subfiles}
\externaldocument[M-]{\subfix{main}}
```

In the `\externaldocument` command, `main` is the name of the main document. Moreover, `M-` is an arbitrary sequence of characters that is added as prefix to the labels. The `\subfix` command is only needed if the subfiles are not in the same directory as the main file, but it doesn't hurt if you add it in any case.

To cross-reference between documents, add labels as usual. Suppose you have `\label{mylabel}` in any of the files. Then you can use `\ref{M-mylabel}` and `\pageref{M-mylabel}` to obtain the (page) number that the label refers to in the main document.

Note that you first have to compile `main.tex`. This generates `main.aux`, which then can be loaded by the subfiles to provide the information for the labels prefixed with `M-`.

4.3 Avoiding extra spaces

Sometimes you may want to load the contents of a subfile without white space separating it from the contents of the main file. In this respect, `\subfile` behaves similar to `\input`. Any space or newline before and after the `\subfile` command will appear in the typeset document, as will any white space between the last character of the subfile and `\end{document}`. Therefore, to load the contents of a subfile without intervening spaces, you have either to add comment signs:

`main.tex`	`sub.tex`
`...`	`\documentclass[main.tex]{subfiles}`
`text before%`	`\begin{document}`
`\subfile{sub.tex}%`	`contents of subfile%`
`text after`	`\end{document}`

or to put everything on the same line:

```
text before\subfile{sub.tex}text after
   contents of subfile\end{document}
```

5 Troubleshooting

Here are some hints that solve most problems.

1. Make sure to use the most recent version of the `subfiles` package, available from CTAN[1] and Github[2].
2. Make sure that `\usepackage{subfiles}` appears near the end of the main preamble.
3. Make sure that the strings `\begin{document}` and `\end{document}` appear on lines of their own and that there are no additional characters preceding or trailing them.
4. If some external program that cooperates with TeX, like `bibtex` or `biber`, complains about not being able to find a file, locate the name of the file in the LaTeX source and replace ⟨*filename*⟩ by `\subfix{`⟨*filename*⟩`}`.
5. If nothing of the above helps, ask the nice people on tex.stackexchange[3].

6 Dependencies

The `subfiles` package uses the `import` package by Donald Arsenau to load subfiles from different directories. `import.sty` is part of the standard TeX distribution.

7 Version history

v1.1: Initial version by Federico Garcia. Subsequent versions by Gernot Salzer.

v1.2:

- Incompatibility with classes and packages removed that modify the `\document` command, like the class `revtex4`.

v1.3:

- Use of `import` package to handle directory hierarchies.
- `\ignorespaces` added to avoid spurious spaces.
- Incompatibility with commands removed that expect `\document` to be equal to `\@onlypreamble` after the preamble. Thanks to Eric Domenjoud for analysing the problem.

v1.4:

- Incompatibility with `memoir` class and `comment` package removed.
- Bug '`\unskip` cannot be used in vertical mode' fixed.

v1.5:

[1] `https://ctan.org/pkg/subfiles`
[2] `https://github.com/gsalzer/subfiles`
[3] `https://tex.stackexchange.com/`

- Command `\subfileinclude` added.
- Basic support for `bibtex` related bibliographies in subfiles added. Seems to suffice also for sub-bibliographies with the package `chapterbib`.
- Support for sub-bibliographies with package `bibunits` added.

v1.6:

- Support for sub-bibliographies with package `bibunits` dropped, in favor of `\subfix`.
- Command `\subfix` added.
- Incompatibility with `standalone` class removed.
- The options of the main class are now also processed when typesetting a subfile; before they were ignored. Thanks to Ján Kl'uka for analysing the problem.

v2.0:

- Incompatibility with LaTeX Oct. 2020 removed. Thanks to Ulrike Fischer from the LaTeX 3 team for the warning in time.
- By default, text after `\end{document}` as well as the preamble of subfiles, when loaded with `\subfile`, are ignored now. The old behaviour is available via the new package option `v1`.
- Command `\ifSubfilesClassLoaded` added and documentation regarding the use of the `\bibliography` command corrected. Thanks to Github user `alan-isaac` for reporting the issue.
- Subfiles now can have the same name as the main file. Thanks to Github user `June-6th` for reporting the issue.
- Problem with the search path for images resolved. Thanks to Github user `maxnick` for reporting the issue.
- Section about cross-referencing added. Thanks to Github user `ndvanforeest` for the input.

8 The Implementation

8.1 The class

```
⟨*class⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{subfiles}[2020/09/07 v2.0 Multi-file projects (class)]
\DeclareOption*{%
  \typeout{Preamble taken from file `\CurrentOption'}%
  \let\preamble@file\CurrentOption
}
\ProcessOptions
```

After processing the option of the `subfiles` class, we reset `\@classoptionslist` such that the options in the main file will be processed.

```
\let\@classoptionslist\relax
```

To handle subfiles in separate directories, we use the `import` package. We load it now, since it resets the macro `\import@path`.

```
\RequirePackage{import}
```

We redefine `\documentclass` to load the class of the main document.

```
\let\subfiles@documentclass\documentclass
\def\documentclass{%
  \let\documentclass\subfiles@documentclass
  \LoadClass
}
```

In earlier versions, we used `\subimport` to load the preamble of the main file, which has the unwanted effect of undoing changes to the graphics path. Therefore we use `\input` and initialize `\import@path` and `\input@path` to the path of the main file. We use the internal LaTeX macro `\filename@parse` to obtain this path.

```
\filename@parse{\preamble@file}
\edef\import@path{\filename@area}
\edef\input@path{{\filename@area}}
\input{\preamble@file}
```

After loading the preamble of the main file, we reset `\import@path{}`. Since the preamble may have changed the catcode of the `@` sign, we make it (again) a letter. Better safe than sorry.

```
{\makeatletter
  \gdef\import@path{}
}
⟨/class⟩
```

8.2 The package

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{subfiles}[2020/09/07 v2.0 Multi-file projects (package)]
```

The package has one option, `v1`, which affects the way how the text after `\end{document}` and in the preamble of subfiles is handled. With this option, `subfiles` behaves like in version 1.x. To implement the retro behaviour, we need a few definitions.

First, we define three macros containing the strings `\begin{document}`, `\end{document}` and `document`. We need them later to detect these strings in the input.

```
\def\subfiles@DOCUMENT{document}
{\escapechar=-1\relax
 \xdef\subfiles@BEGINDOCUMENT{\string\\begin\string\{document\string\}}%
 \xdef\subfiles@ENDDOCUMENT{\string\\end\string\{document\string\}}%
}
```

The macro `\subfiles@skipDocument` skips everything until it encounters the string `\end{document}`. In earlier versions of the package, this was accomplished by redefining the `document` environment to the `comment` environment of the `verbatim` package. This clashes with the LaTeX format published in October 2020, so we copy and adapt the code from the `verbatim` package (see the definition of `\subfiles@skiplines` below).

```
\def\subfiles@skipDocument{%
  \let\subfiles@skiplinescont\ignorespaces
  \let\subfiles@skiplinesend\subfiles@ENDDOCUMENT
  \subfiles@skiplines
}
```

To skip `\documentclass` the old way, we redefine it to do nothing except restoring its original definition.

```
\def\subfiles@skipDocumentclass{%
  \renewcommand\documentclass[2][]{%
    \let\documentclass\subfiles@documentclass
    \ignorespaces
  }%
}
```

To ignore the preamble of a subfile, we redefine `\documentclass` to skip everything until the string `\begin{document}` is encountered.

```
\def\subfiles@skipPreamble{%
  \def\documentclass{%
    \let\documentclass\subfiles@documentclass
    \def\subfiles@skiplinescont{\begin{document}}%
    \let\subfiles@skiplinesend\subfiles@BEGINDOCUMENT
    \subfiles@skiplines
  }%
}
```

Now we set the default behaviour of the subfiles package: When loading the main preamble in a subfile, everything after the preamble is ignored. Moreover, when reading a subfile, its preamble as well as everything after `\end{document}` is ignored.

```
\let\subfiles@handleMain\endinput
\let\subfiles@handleSubpreamble\subfiles@skipPreamble
\let\subfiles@handleTextAfterSubdocument\endinput
```

The option `v1` restores the behaviour of the old `subfiles` package: When loading the main preamble in a subfile, only the contents of the `document` environent is ignored, but not the stuff following it. Moreover, when reading a subfile, only the `\documentclass` command and the lines `\begin{document}` and `\end{document}` are ignored, but the subfile preamble as well as everything after `\end{document}` is retained.

```
\DeclareOption{v1}{%
  \let\subfiles@handleMain\subfiles@skipDocument
  \let\subfiles@handleSubpreamble\subfiles@skipDocumentclass
  \let\subfiles@handleTextAfterSubdocument\relax
}
```

```
\DeclareOption*{\PackageWarning{subfiles}{Option '\CurrentOption' ignored}}
\ProcessOptions\relax
```

To skip everything until a specific string is read, we adapt code from the `verbatim` package. The skipping of lines is controlled by two macros that have to be set before calling `\subfiles@skiplines`. `\subfiles@skiplinesend` is the string that marks the end of the skipped area; it has to appear on a line of its own, as the only content of this line. `\subfiles@skiplinescont` contains the code to be executed after skipping has ended.

```
\def\subfiles@skiplines{%
  \begingroup
    \let\do\@makeother\dospecials
    \@makeother\^^L%
    \endlinechar`\^^M\relax \catcode`\^^M=12\relax \subfiles@skipline}
{\catcode`\^^M=12 \endlinechar=-1 %
 \gdef\subfiles@skipline#1^^M{\def\subfiles@tmp{#1}%
      \ifx\subfiles@tmp\subfiles@skiplinesend
          \def\subfiles@tmp{\endgroup\subfiles@skiplinescont}%
      \else\let\subfiles@tmp\subfiles@skipline
      \fi \subfiles@tmp}
}
```

To handle subfiles in separate directories, we use the `import` package. If it has already been loaded, e.g. by the `subfiles` class, this line does nothing.

```
\RequirePackage{import}
```

The `\subimport` command requires path and filename as separate arguments, so we have to split file locations into these two components. The internal LaTeX command `\filename@parse` almost fits the bill, except that it additionally splits the filename into basename and extension. Unfortunately, concatenating basename and extension to recover the filename is not clean: Under Unix/Linux, the filenames `base` and `base.` denote different entities, but after `\filename@parse` both have the same basename and an empty extension. Therefore we redefine the command `\filename@simple` temporarily; it is responsible for this unwanted split.

```
\def\subfiles@split#1{%
  \let\subfiles@filename@simple\filename@simple
  \def\filename@simple##1.\\{\edef\filename@base{##1}}%
  \filename@parse{#1}%
  \let\filename@simple\subfiles@filename@simple
}%
```

E.g., after executing `\subfiles@split{../dir1/dir2/file.tex}` the macros `\filename@area` and `\filename@base` expand to `../dir1/dir2/` and `file.tex`, respectively.

`\subfile` The command `\subfile` specifies the command `\subimport` for `\input`ing the subfile, and then calls `\subfiles@subfile`.

```
\newcommand\subfile{%
  \let\subfiles@loadfile\subimport
  \subfiles@subfile
}
```

`\subfileinclude` The command `\subfileinclude` specifies the command `\subincludefrom` for `\include`ing the subfile, and then calls `\subfiles@subfile`.

```
\newcommand\subfileinclude{%
  \let\subfiles@loadfile\subincludefrom
  \subfiles@subfile
}
```

The main functionality of the two commands is implemented in `\subfiles@subfile`. It redefines `\documentclass` and the `document` environment to do nothing but reverting these command to their original meaning and avoiding spurious spaces. Reverting `\documentclass` and `\document` to their original definition is important for being compatible with classes like `standalone` or packages like `bibentry`, which rely on this definition.

```
\newcommand\subfiles@subfile[1]{%
  \begingroup
  \let\subfiles@documentclass\documentclass
  \let\subfiles@document\document
  \let\subfiles@enddocument\enddocument
  \subfiles@handleSubpreamble
  \renewenvironment{document}{%
    \let\document\subfiles@document
    \ignorespaces
  }{%
    \let\enddocument\subfiles@enddocument
    \@ignoretrue
    \subfiles@handleTextAfterSubdocument
  }%
```

Now we split the file name into path and base name and load the file.

```
  \subfiles@split{#1}%
  \subfiles@loadfile{\filename@area}{\filename@base}%
  \endgroup
}
```

`\subfix` If some package provides a command that takes a filename as argument, then it has to be prefixed with the current `\import@path`. This is what the `\subfix` command tries to do. In order to succeed, the filename has to be expanded immediately, such that the current value of `\import@path` is used.

```
\def\subfix#1{\import@path#1}
```

For patching a list of file or path names, we define two auxiliary macros, one iterating over a comma-separated list of names and one processing a sequence of names enclosed in braces.

```
\def\subfiles@fixfilelist#1{%
  \def\subfiles@list{}%
  \def\subfiles@sep{}%
  \@for\subfiles@tmp:=#1\do{%
    \edef\subfiles@list{\subfiles@list\subfiles@sep\subfix{\subfiles@tmp}}%
    \def\subfiles@sep{,}%
  }%
}
\def\subfiles@fixpathlist#1{%
  \def\subfiles@list{}%
  \@tfor\subfiles@tmp:=#1\do{%
    \edef\subfiles@list{\subfiles@list{\subfix\subfiles@tmp}}%
  }%
}
```

\bibliography
\graphicspath

We patch `\bibliography` and `\graphicspath` (from the `graphics`/`graphicx` package) such that users don't have to worry about adding `\subfix`.

```
\let\subfiles@bibliography\bibliography
\renewcommand\bibliography[1]{%
  \subfiles@fixfilelist{#1}%
  \expandafter\subfiles@bibliography\expandafter{\subfiles@list}%
}
\@ifpackageloaded{graphics}{%
  \let\subfiles@graphicspath\graphicspath
  \renewcommand\graphicspath[1]{%
    \subfiles@fixpathlist{#1}%
    \edef\subfiles@list{{\subfix{}}\subfiles@list}%
    \expandafter\subfiles@graphicspath\expandafter{\subfiles@list}%
  }%
}{}
```

\ifSubfilesClassLoaded

To add code or text conditionally, depending on whether the main document or a subfile is typeset, we provide the command `\ifSubfilesClassLoaded`.

```
\newcommand\ifSubfilesClassLoaded{%
  \expandafter\ifx\csname ver@subfiles.cls\endcsname\relax
    \expandafter\@secondoftwo
  \else
    \expandafter\@firstoftwo
  \fi
}
```

The `subfiles` package is loaded near the end of the main preamble. If it is loaded from a subfile, i.e., if `subfiles.cls` has been loaded, then we have to prepare for skipping the main document. We do this be redefining the `\begin` command. Normally, the first `\begin` after the `subfiles` package starts the main document. To allow for the case that some other environment occurs before (does it really happen?) we test whether we are dealing with `\begin{document}`. If not, we execute the original definition of `\begin`; otherwise we skip the main document as specified by `\subfiles@handleMain`.

```
\ifSubfilesClassLoaded{%
  \let\subfiles@begin\begin
  \def\begin#1{%
    \def\subfiles@tmp{#1}%
    \ifx\subfiles@tmp\subfiles@DOCUMENT
      \let\begin\subfiles@begin
      \let\subfiles@tmp\subfiles@handleMain
    \else
      \def\subfiles@tmp{\subfiles@begin{#1}}%
    \fi
    \subfiles@tmp
  }%
}{}
⟨/package⟩
```