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

Federico Garcia, Gernot Salzer

2019/11/06 v1.6

Abstract

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

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` *at the end of the preamble*:

```
\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 three lines `\documentclass[..]{subfiles}`, `\begin{document}`, and `\end{document}` are ignored.

3 Advanced usage

3.1 Hierarchy of directories

Sometimes it is desirable to put a subfile together with its images and further 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 files 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.

3.2 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*⟩`}`

3.3 Bibliographies

Manual bibliographies with the `thebibliography` environment work as usual. Problems may arise if external programs like `bibtex` or `biber` are used to generate the bibliography. Here are some hints on how to make it work.

\bibliography

- Make sure the command `\bibliography` is executed after loading the `subfiles` package. Put the command between `\usepackage{subfiles}` and `\begin{document}` or somewhere into the text part.
- When you use the package `biblatex`, and programs like `biber` complain about not being able to find the bibliography files, use `\bibliography` instead of `\addbibresource` (see above), or the command `\subfix` (see below).

\subfix

- Whenever an external program complains that a file specified in the LaTeX document cannot be found, wrap the command `\subfix` around the filename. Here are some examples.

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

The `subfiles` package has been tested with the packages `biblatex`, `bibunits`, and `chapterbib` as well as with the external programs `bibtex` and `biber`.

3.4 Unusual locations for placing definitions and text

Usually all definitions and packages required by the subfiles should go into the preamble of the main file. There are some further locations, though, where one might consider adding definitions and text. Put negatively, apparently irrelevant stuff at these locations may become unexpectedly visible in the document or causes errors.

Code after the end of the main document is added to the preamble of the subfiles, but is ignored when typesetting the main file. This happens because a subfile typeset by itself does not really take the preamble of the main file, but *everything outside* of `\begin{document}` and `\end{document}`. This has two consequences: *a)* the user can add some commands to be processed as part of the preamble only when the subfiles are typeset by themselves; but also *b)* the user has to be careful even *after* `\end{document}` in the main file, for any syntax error there will ruin the LaTeXing of the subfile(s).

Similarly, when typesetting the main document, the `\subfile` command does not really load the stuff within the `document` environment, but *everything except* the three commands `\documentclass[...]{...}`, `\begin{document}`, and `\end{document}`. This has the following consequences.

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 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 within 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.

3.5 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}`. Moreover, any stuff after `\end{document}` will end up in the main document, including spurious empty lines, which may lead to a new paragraph. Therefore, to load the contents of a subfile without intervening spaces, you have either to add comment signs:

main.tex

```
...
text before%
\subfile{sub.tex}%
text after
```

sub.tex

```
\documentclass[main.tex]{subfiles}
\begin{document}
contents of subfile%
\end{document}
% No empty lines after \end{document}!
```

or to put everything on the same line:

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

4 Dependencies

The `subfiles` package requires the `verbatim` package, whose `comment` environment is used to ignore the text area of the main file when typesetting subfiles separately. Moreover, the `import` package is needed to load subfiles and their auxiliary files from different directories. Both packages are part of the standard TEX distributions.

5 Version history

v1.1: Initial version by Federico Garcia. Further 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:

- 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.

6 The Implementation

6.1 The class

```
1 ⟨*class⟩
2 \NeedsTeXFormat{LaTeX2e}
3 \ProvidesClass{subfiles}[2019/11/06 v1.6 Multi-file projects (class)]
4 \DeclareOption*{\typeout{Preamble taken from file `\CurrentOption'}%
5     \let\preamble@file\CurrentOption}
6 \ProcessOptions
```

We start by saving the regular LaTeX definition of `\documentclass`:

```
7 \let\subfiles@documentclass\documentclass
```

Now `\documentclass` is set equal to `\LoadClass` such that the class and the options of the main file will be loaded as usual.

```
8 \let\documentclass\LoadClass\relax
```

When typesetting a subfile, we have to skip the `document` environment of the main file. This is done with the commands `\comment` and `\endcomment` from the `verbatim` package. Now there is a problem: If we load `verbatim` here, the

definition of the commands may be overwritten if the user loads e.g. the `comment` package. Loading `verbatim` in `subfiles.sty` at the latest possible moment is not reliable, either. On the one hand we may overwrite macros required later by the user, on the other hand the `memoir` class contains a copy of `verbatim`, so a later `\RequirePackage` refuses to reload the package. Thus, in the case of a document loading the `memoir` class and the `comment` package, we end up with the wrong definition of `\comment` in any case.

Therefore we load the `verbatim` package here and save the contents of the crucial commands `\comment` and `\endcomment` under a different name.

```
\RequirePackage{verbatim}
\let\subfiles@comment\comment
\let\subfiles@endcomment\endcomment
```

To handle subfiles in separate directories, we load the `import` package.

```
\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.

Now we split the name of the main file that has been provided as optional argument of the document class, and `\subimport` the main file.

```
\subfiles@split{\preamble@file}
\subimport{\filename@area}{\filename@base}
```

The main file loads the package `subfiles` as part of the preamble, which saves the contents of `\document` and `\enddocument` as `\subfiles@document` and `\subfiles@enddocument`, respectively. We use these macors now to restore the original values of `\document`, `\enddocument`, and `\documentclass`. The backup commands are `\undefined` to save memory. That's it.

```
{\catcode`\@=11
\global\let\document\subfiles@document
\global\let\enddocument\subfiles@enddocument
\global\let\documentclass\subfiles@documentclass
\global\let\subfiles@document\undefined
```

```
\global\let\subfiles@enddocument\undefined
\global\let\subfiles@documentclass\undefined
}
```
⟨/class⟩

It may not be obvious why `@` has to be catcoded to a letter, since we are in a style file anyway. However, the `\preamble@file` occasionally contains `\usepackage` commands that make `@` a non-letter. This is why the part after loading the main preamble needs a `\catcode` command, grouping, and `\global`'s.

6.2 The package

Any option will be ignored.

⟨*package⟩
```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{subfiles}[2019/11/06 v1.6 Multi-file projects (package)]
\DeclareOption*{\PackageWarning{\CurrentOption ignored}}
\ProcessOptions
```

If the initial document class was `subfiles`, then the main file is loaded as part of a subfile. In this case anything between `\begin{document}` and `\end{document}` has to be skipped, while the contents of the commands `\document` and `\enddocument` has to be retained for later use in the subfile. Therefore we save the contents of the two commands as `\subfiles@document` and `\subfiles@enddocument`, respectively. Now the `document` environment is redefined to become the saved `comment` environment from the `verbatim` package. Consequently, the body of the main file is ignored by LaTeX, and only the preamble is read (as well as anything that comes after `\end{document}`!).

```
\@ifclassloaded{subfiles}{%
  \let\subfiles@document\document
  \let\subfiles@enddocument\enddocument
  \let\document\subfiles@comment
  \let\enddocument\subfiles@endcomment
```

By loading the `subfiles` package immediately before `\begin{document}` we ensure that `\subfiles@document` and `\subfiles@enddocument` contain all modifications that the class and the preamble of the main file may have applied to the `document` environment. This happens e.g. with the class `revtex4` and the package `pythontex`.

We use the `import` package to handle subfiles in separate directories. 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. Both things, loading the package and defining the command, are also done in

`subfiles.cls`, so we have to execute this code only if we are typesetting the main file.

```
}{% subfiles class not loaded, we typeset the main document
  \RequirePackage{import}
  \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 `\subfile` 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. Returning `\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
  \renewcommand\documentclass[2][subfiles]{%
    \let\documentclass\subfiles@documentclass
    \ignorespaces
  }%
  \renewenvironment{document}{%
    \let\document\subfiles@document
    \ignorespaces
  }{%
    \let\enddocument\subfiles@enddocument
```

```
    \@ignoretrue
  }%
```

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
}
```

To let external programs find files, we have to add the `\import@path` to file names. This is accomplished with the command `\subfiles@addimportpath`.

```
\def\subfiles@addimportpath#1{%
  \def\subfiles@filelist{}%
  \def\subfiles@sep{}%
  \@for\subfiles@filename:=#1\do{%
    \edef\subfiles@filelist{%
      \subfiles@filelist
      \subfiles@sep
      \import@path
      \subfiles@filename
    }%
    \def\subfiles@sep{,}%
  }
}
```

`\bibliography` We redefine the `\bibliography` command such that the import path is added to the file names before the original command is called.

```
\let\subfiles@bibliography\bibliography
\renewcommand\bibliography[1]{%
  \subfiles@addimportpath{#1}%
  \expandafter\subfiles@bibliography\expandafter{\subfiles@filelist}%
}
```

`\subfix` Instead of adding further fixes for other packages that write filenames to external files (like `bibunits`), we provide a command for adding the `\import@path` to a filename.

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