The childdoc Package

Niklas Beisert

Institut für Theoretische Physik
Eidgenössische Technische Hochschule Zürich
Wolfgang-Pauli-Strasse 27, 8093 Zürich, Switzerland

nbeisert@itp.phys.ethz.ch

21 May 2017, v1.5

Abstract

childdoc is a LaTeX 2_ε package that enables the direct compilation of document sections included by `\include` to individual files.

Contents

1 Introduction

LaTeX provides a mechanism to structure a large document (such as a book) into a main file and several child files (containing the chapters) using the `\include` command. This mechanism is beneficial for documents which span hundreds of pages in order to make the source file(s) more manageable. Moreover, compilation can be restricted to selected child files by means of the `\includeonly` command. The latter feature can be used to reduce the compilation time while editing (this was significantly more useful in the earlier days of LaTeX) or to generate a smaller document which is easier to navigate. Another application of `\includeonly` is to generate documents consisting of selected parts of the complete document.

However, there are a few drawbacks of the plain `\include` mechanism:

- The child files cannot be compiled on their own, they can only be compiled via the main file. A naive editing environment (such as a text editor with an option to have the current file processed by LaTeX) may require one to switch to the main file before compiling; attempting to compile the child file produces errors.
- The main file must be modified (each time) to adjust the `\includeonly` command to the present needs. This easily leaves the main file in a messy state.
- The generated document will always carry the filename of the main document. This is inconvenient if several child files are to be compiled and to be kept for distribution.

The present package provides a simple interface to make child files individually compilable by LaTeX. Compiling a child file then has the same effect as compiling the main file with an `\includeonly` command to select the appropriate child. Moreover the generated document will carry the name of the child rather than the main file. This resolves all three above issues.

This feature is meant to make the editing of books, thesis documents and lecture notes somewhat more convenient. However, the package can also be used efficiently for composing a series of documents (such as exercise sheets) which are typically distributed individually. It then assists the author in generating the individual documents (potentially in different versions) as well as a document containing the collected series.

2 Usage

The package childdoc is *not* a standard LaTeX 2_ε `.sty` style file! Therefore it needs to be invoked in a non-standard way.

`\childdoc` To use the package add the commands

```
\input{childdoc.def}
\childdocmain{main}
```

at the very top of your main LaTeX file, in particular *before* the `\documentclass` statement! Furthermore, you must add the command

```
\input{childdoc.def}
\childdocof{main}
```

at the top of every child file which is included by `\include` from within the main file (or at least for those files which you would like to compile individually). In each case, the argument *main* must be the filename of the main file. Alternatively, it suffices to start a child file with just `\input{`*main*`}`; this has the same effect, but the nesting of included files is slightly different. Note that the closed loop generated by this combination of `\input` and `\include` commands is broken by `\childdocmain`.

Please note the following restrictions:

- The argument *main* of `\childdocmain` must match the filename of the file in which it is specified. This is necessary because TeX does not store the name of files included via `\input`.
- The filename *main* must be specified without the `.tex` extension.
- The filename *main* is case sensitive (even in case-insensitive file systems) due to internal string comparison.
- The argument *main* should be fully expanded, it cannot be a macro.
- Subdirectories and special characters should be avoided in filenames.

- The command `\childdocmain{`*main*`}` must be followed by a whitespace. It should not be followed immediately by another command or by a comment mark '`%`'. This is because the TEX parser reads the token immediately following the argument of `\childdocmain` and puts it at the beginning of every child section; however, a whitespace is ignored.

2.1 Considerations

Let us mentions a couple of consideration in setting up the main and child documents:

Content of Main File. It is advisable to place all output in the child files included by `\include`. Any output contained in the main file will appear in all child documents; it cannot be suppressed by the `\includeonly` directive and thus should normally be avoided. Below we shall describe a method to include some output in the main file by means of conditional processing.

Page Numbering. When only a part of the document is compiled, the appropriate numbering of pages (as well as other status parameters) is determined from the `.aux` files. The latter contain information from previous passes. However this information needs to propagate through all intermediate child documents. Therefore the page numbering in child documents may well be inconsistent until the complete document is compiled at least once.

A useful (if unconventional) way to always ensure a consistent page numbering is to restart the numbering in each child document and denote the pages by '*child*`.`*page*' where *child* represents the chapter/section number of the child file. This can be achieved by the command `\numberwithin{page}{`*child*`}` of the `amsmath` package where *child* can be `chapter` or `section` depending on the chosen structuring. Alternatively, one can modify the macro `\thepage` appropriately and reset the counter `page` at the start of each child file.

2.2 Conditional Processing

The package provides a mechanism to compile different versions of a document. To customise the versions further some conditional processing can come in handy to distinguish which version is being compiled. The package provides two macros to describe the compilation context:

`\ifchilddoc` The conditional `\ifchilddoc` distinguishes between the compilation of child documents and the main document:

`\ifchilddoc` *child-code* [`\else` *main-code*] `\fi`

`\childdocname` The macro `\childdocname` contains the filename (without extension) of the main or child file being processed. Note that `\jobname` will always contain the name of the main file.

Title Page. For example, conditional processing can be used to include a title or banner page in the main document when proper precautions are taken. Importantly, the code in the main file should ensure that the page counter (as well as other status parameters which are stored in the `.aux` files) takes the same value after the conditional processing. Otherwise the page numbers may take divergent values depending on which part is compiled.

For example, a title page could be declared by:

```
\ifchilddoc\else
\addtocounter{page}{-1}
code for title page
\newpage
\fi
```

A banner page for the child documents can be generated by:

```
\ifchilddoc
\addtocounter{page}{-1}
code for banner page
\newpage
\fi
```

Here one could write a message such as:

```
This is the part \childdocname{} of \jobname.
```

2.3 Flags and Forwarding

The package allows to easily generate different versions of the main or child documents and to (permanently) store these in different files. To this end compilation flags can be defined and assigned different default values.

Defining Flags. Suppose we want to define a flag `\version` which can be set to `draft` or `final`. The document source will contain some conditional code depending on the value of `\version`. Suppose further, the flag should default to `final` for the main file and to `draft` for child files which is a natural assignment for editing the document. This is achieved by placing the following code in the preamble of the main document (below the `\childdocmain` directive):

```
\ifchilddoc
\providecommand{\version}{draft}
\else
\providecommand{\version}{final}
\fi
```

By using `\providecommand` we make sure that previous definitions are not overwritten. We can thus add further statements `\providecommand{\version}{...}` before the above code to override it.

For the main file, one might add a line (between `\childdocmain` and the above block)

```
%\ifchilddoc\else\providecommand{\version}{draft}\fi
```

which can be uncommented to produce a draft version. Likewise one can add a line to the very top of a child file (above the `\childdocof{`*main*`}` directive)

```
%\providecommand{\version}{final}
```

which can be uncommented to produce the final version of this child document.

Forwarding. Once compilation flags are defined, we can permanently set up files to produce certain versions of the documents. To this end, the package defines a command to pass on compilation to a different file:

`\childdocforward` `ilddocforwardprefix`

The command `\childdocredirect` redirects processing to the another source file:

```
\childdocforward[main]{dest}
\childdocforwardprefix[main]{prefix}{dest}
```

The argument *dest* is the destination file (without extension). It should be the main file or one of the child files. In the second form, the destination file is determined by a pattern depending on the current file: To make this work, the current file must be called '*prefix suffix*' and processing is passed on to the file '*dest suffix*'. Surely, the same effect is achieved

by directly specifying the argument '*dest suffix*' in the first form. However, that requires to set up a different file for each child. With the alternative form of the command all these files can have exactly the same content which simplifies setting them up and maintaining them. Finally, the optional argument *main* allows to pass on directly to the main file *main* while pretending to compile *dest*.

For example, the following file `draft.tex` compiles the main document as a draft:

```
\def\version{draft}
\input{childdoc.def}
\childdocredirect{main}
```

Likewise, the following files `final`*nn*`.tex` compile the final version of the child document `child`*nn*`.tex`:

```
\def\version{final}
\input{childdoc.def}
\childdocredirect{final}{child}
```

Note that when several versions of a main file and/or of each child file are to be generated, it will be convenient to set up a `Makefile` or shell script to automatise the process.

2.4 Manual Code

In case one cannot be certain whether the definitions file `childdoc.def` is installed on the target TEX distribution and one prefers not to ship it, it is conceivable to paste a few relevant commands into the sources.

To that end, drop all statements `\input{childdoc.def}` and perform the replacements as outlined below. Instead of `\childdocmain{`*main*`}` add the following code to the top of the main file:

```
\ifdefined\childdocname\endinput\fi\newif\ifchilddoc
\edef\childdocname{\scantokens\expandafter{\jobname\noexpand}}
\def\childdocmain{main}\ifx\childdocmain\childdocname\else
\childdoctrue\includeonly{\childdocname}\let\jobname\childdocmain\fi
```

Instead of `\childdocof{`*main*`}` just include the main file at the top of each child file:

```
\input{main}
```

A simple redirection `\childdocforward{`*dest*`}` is achieved by:

```
\def\jobname{dest}\input{\jobname}
```

The redirection with prefix `\childdocforwardprefix[`*prefix*`]{`*dest*`}` is accomplished by:

```
{\edef\jobname{\scantokens\expandafter{\jobname\noexpand}}
\def\redirectjob prefix#1~~~{\gdef\jobname{dest#1}}
\expandafter\redirectjob\jobname~~~}\input{\jobname}
```

2.5 Command Line Processing

The effect of redirection files can also be achieved by invoking the LATEX compiler with a more elaborate command line. Most conveniently this should be done as part of a shell script or a `Makefile`.

When using childdoc in the main file, the following command line effectively performs a redirection (note that depending on the shell being used, backslashes may have to be doubled: '\' → '\\')

```
... -jobname "target" "[flags]\def\jobname{dest}\input{main}"
```

Here *target* is the name of the output file, *main* is the name of the main file and *dest* is the name of the main or child file to be processed (all filenames without extensions). Optionally, compilation *flags* can be defined via `\def` commands.

This command line makes the TEX engine believe it is compiling the file *target* whose content is specified as the latter parameter. The provided code in turn tweaks the definition of `\jobname` to *dest* which is later passed on to `\includeonly` by `\childdocmain` and then hands over to the main file *main*.

In fact, a similar effect can be achieved without the childdoc mechanism by using the command line:

```
... -jobname "target" "[flags]\includeonly{dest}\input{main}"
```

However, some of the functionality of childdoc is lost, e.g. child documents cannot be processed individually and the conditional `\ifchilddoc` is not defined.

3 Information

3.1 Copyright

3.2 Files and Installation

The package consists of the files:

`README`	readme file
`childdoc.ins`	installation file
`childdoc.dtx`	source file
`childdoc.def`	definition file
`cdocsamp.tex`	sample main file
`cdocsch1.tex`	sample include file
`cdocsch2.tex`	sample include file
`cdocsdrf.tex`	sample redirection file
`cdocsfn1.tex`	sample redirection file
`cdocsfn2.tex`	sample redirection file
`childdoc.pdf`	manual

The distribution consists of the files `README`, `childdoc.ins` and `childdoc.dtx`.

- Run (pdf)LATEX on `childdoc.dtx` to compile the manual `childdoc.pdf` (this file).
- Run LATEX on `childdoc.ins` to create the definitions file `childdoc.def` and the sample `cdocsamp.tex` with include files `cdocsch1.tex`, `cdocsch2.tex`, `cdocsdrf.tex`, `cdocsfn1.tex`, `cdocsfn2.tex`. Copy the file `childdoc.def` to an appropriate directory of your LATEX distribution, e.g. *texmf-root*`/tex/latex/childdoc`.

3.3 Related CTAN Packages

There are several other packages which offer a similar functionality:

- The packages docmute, includex and standalone provide commands to include only the document body of a child file thus allowing both files to be compiled individually.
- The packages subdocs and subfiles provide structures in which the main and child documents can be encapsulated and allowing them to be compiled individually. The inclusion mechanism is different from the conventional `\include`.
- The package combine is an elaborate solution to combine several documents into one.

See also the CTAN topic subdocs for further related packages. The present package differs from the above solutions in that a document structure constructed with the conventional `\include` mechanism just needs two extra commands at the top of every file such that all constituent files can be compiled individually.

3.4 Revision History

v1.5: 2017/05/21

- more complete structuring introduced
- `\childdocof` introduced
- `\childdoc` renamed to `\childdocmain`
- `\childredirect` renamed to `\childdocforward` and `\childdocforwardprefix` and functionality expanded

v1.0: 2017/04/27

- manual and install package
- first version published on CTAN

v0.6: 2017/04/26

- redirection mechanism added

v0.5: 2016/04/26

- functionality in definition file

A Sample

Here we present a sample document with two chapters, a title page, a compile flag as well as three forwarding files to set the flag. It consists of six `.tex` files:

`cdocsamp.tex`	main file
`cdocsch1.tex`	include file for chapter 1
`cdocsch2.tex`	include file for chapter 2
`cdocsdrf.tex`	forwarding file for main file in draft mode
`cdocsfi1.tex`	forwarding file for final version of chapter 1
`cdocsfi2.tex`	forwarding file for final version of chapter 2

Each of the six files can be compiled directly by the LaTeX compiler.

Main File. The main file is called `cdocsamp.tex`.

Load the childdoc definitions and declare the filename for the main document:

```
\input{childdoc.def}
\childdocmain{cdocsamp}
```

Optional override for `\version` flag:

```
%%\ifchilddoc\else\providecommand{\version}{draft}\fi
```

Define the default values for the `\version` flag (`final` for the main file and `draft` for childs):

```
\ifchilddoc
\providecommand{\version}{draft}
\else
\providecommand{\version}{final}
\fi
```

Load the standard document class:

```
\documentclass[12pt]{article}
```

Start the document body:

```
\begin{document}
```

Declare a title page. Print title, part of document being processed and version flag:

```
\addtocounter{page}{-1}
\begin{center}
{\LARGE\bfseries{}childdoc example\par}
\vspace{1cm}
part:
\ifchilddoc
‘\childdocname’ of ‘\jobname’\par
\else
main\par
\fi
version: \version\par
\end{center}
\newpage
```

Include the two chapters:

```
\include{cdocsch1}
\include{cdocsch2}
```

End of document body:

```
\end{document}
```

Chapter Include Files. The chapter include files are called `cdocsch1.tex` and `cdocsch2.tex`.

Optional override for `\version` flag:

```
%%\providecommand{\version}{final}
```

Include the main document:

```
\input{childdoc.def}
\childdocof{cdocsamp}
```

Some text for chapter 1:

```
\section{one}
some text in chapter one
```

Some text for chapter 2:

```
\section{two}
more text in chapter two
```

Forwarding for a Complete Draft. The following forwarding file `cdocsdrf.tex` compiles the main document in draft mode:

```
\def\version{draft}
\input{childdoc.def}
\childdocforward{cdocsamp}
```

Forwarding for Final Version of the Chapters. The following forwarding files `cdocsfn1.tex` and `cdocsfn2.tex` (with identical content) compile the final versions of the child documents `cdocsch1.tex` and `cdocsch2.tex`, respectively:

```
\def\version{final}
\input{childdoc.def}
\childdocforwardprefix[cdocsamp]{cdocsfn}{cdocsch}
```

B Implementation

In this section we describe the definitions file `childdoc.def`.

The definitions cannot be loaded using `\usepackage` or `\RequirePackage` which has a mechanism to prevent loading a style file more than once. When loading the definitions by means of `\input` we have to prevent multiple instances manually:

```
%\ifdefined\childdocmain\endinput\fi
```

`\ifchilddoc` The conditional `\ifchilddoc` tells whether a child (true) or main (false) document is being compiled. The definition initialises to false:

```
\newif\ifchilddoc
```

`\childdocname` The macro `\childdocname` stores the name of the document to be compiled. In modern TEX engines the content of `\jobname` appears to be protected to account for special characters or subdirectories in filenames. This prevents a successful comparison to the name of the main file. The following code stores an expanded version of `\jobname` in `\childdocname`:

```
\edef\childdocname{\scantokens\expandafter{\jobname\noexpand}}
```

`\childdocmain` The macro `\childdocmain` is to be called at the top of the main file with the main filename (without extension) as argument. First, it overwrites its own definition to end processing of the present file (`\endinput`) on subsequent calls. It also overwrites the definition of `\childdocof` to prevent further inclusions of the main document. This prevents the main file from being processed more than once. Then the current filename is compared to the main filename and in case of mismatch `\ifchilddoc` is set to true. In that case `\includeonly` is applied to the child file and `\jobname` is set to the main file (for proper handling of `.aux` files):

```
\newcommand{\childdocmain}[1]
{
```

```
  \def\childdocmain##1{\endinput}
  \def\childdocof##1{}
  \begingroup
    \def\childdoctmp{#1}
    \ifx\childdocname\childdoctmp
      \def\childdoctmp{\childdocfalse}
    \else
      \def\childdoctmp{\childdoctrue}
    \fi
    \expandafter
  \endgroup
  \childdoctmp
  \ifchilddoc
    \includeonly{\childdocname}
    \def\jobname{#1}
  \fi
}
```

`\childdoc` The deprecated macro `\childdoc` is a legacy version of `\childdocmain`:

```
\newcommand{\childdoc}{\childdocmain}
```

`\childdocof` The command `\childdocof` redirects compilation to the main file `#1`.

```
\newcommand{\childdocof}[1]
{
  \input{#1}
}
```

`\childdocforward` The command `\childdocforward` redirects compilation to the main or a child file. `\jobname` and `\childdocname` are set to the new filename and compilation is handed over to the new file:

```
\newcommand{\childdocforward}[2][]
{
  \def\jobname{#2}
  \def\childdocname{#2}
  \begingroup
    \def\childdoctmp{#1}
    \def\childdocempty{}
    \ifx\childdoctmp\childdocempty
      \def\childdoctmp{\input{#2}}
    \else
      \def\childdoctmp{\input{#1}}
    \fi
    \expandafter
  \endgroup
  \childdoctmp
  \endinput
}
```

`ilddocforwardprefix` The command `\childdocforwardprefix` redirects compilation to the main or a child file by means of a pattern. The prefix `#1` in the current filename is replaced by `#2` and the suffix of the current filename is kept (it is assumed that the filename does not contain the substring '`~~~`' which is used as a delimiter). Compilation is handed over to the new file by `\childdocforward`:

```
\newcommand{\childdocforwardprefix}[3][]
```

```
{
  \begingroup
    \def\childdocextract #2##1~~~{\def\childdoctmp{\childdocforward[#1]{#3##1}}}
    \expandafter\childdocextract\childdocname~~~
    \expandafter
  \endgroup
  \childdoctmp
}
```

`\childdocredirect` The deprecated command `\childdocredirect` is a legacy version of `\childdocforward` and `\childdocforwardprefix`:

```
\newcommand{\childdocredirect}[2][]
{
  \begingroup
    \def\childdoctmp{#1}
    \def\childdocempty{}
    \ifx\childdoctmp\childdocempty
      \def\childdoctmp{\childdocforward{#2}}
    \else
      \def\childdoctmp{\childdocforwardprefix{#1}{#2}}
    \fi
    \expandafter
  \endgroup
  \childdoctmp
}
```