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

30 December 2018, v2.0

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. Another application is in developing style files or other kinds of included material where compilation of the style file could redirect to a sample or test file.

2 Usage

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

2.1 Included Files

`\childdocmain` To use the package, add the commands

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

at the very top of the main LaTeX file, in particular *before* the `\documentclass` statement! The argument of `\childdocmain` should be left empty (but it must be present).

`\childdocof` Furthermore, add the commands

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

at the top of every child file *child* which is included by `\include{`*child*`}` from within the main file (or at least for those files to be compiled individually). The argument *main* must be the filename of the main file.

There are a couple of considerations in setting up the main and child documents:

Restrictions. Please note the following restrictions:

- `\childdocmain` must be called with one argument *main* to ensure compatibility with earlier version of the package. It must either be empty (`\childdocmain{}`) or precisely match the filename of the main file in which it is specified. See section 2.8 for further information.
- 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.

Content of Main File. It is advisable to place all content in the child files included by `\include`. Any output contained in the main file will appear in all child documents unless suppressed manually; it cannot be suppressed automatically by the `\includeonly` directive and thus should normally be avoided. A method to include some content in the main file by means of conditional processing is described in section 2.2.

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

Title Page. 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 \childdocjob{}.
```

2.3 Flags

The package makes it easy to generate different versions of the main or child documents. To this end compilation flags can be defined and assigned different default values. They will be particularly useful in conjunction with the forwarding mechanism described in section 2.4.

For example, it may be useful to have 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
```

The definition by `\providecommand` makes sure that previous definitions are not overwritten. Further statements `\providecommand{\version}{...}` can thus be added 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.

2.4 Forwarding

Different versions of the main or child documents using compilation flags as described in section 2.3 can be (permanently) stored in different files for convenient compilation, viewing and distribution. To this end, the package defines a command to pass on compilation to a different file:

`\childdocforward` The command `\childdocforward` redirects processing to another source file:

```
\input{childdoc.def}
\childdocforward[main]{dest}
```

The argument *dest* is the destination file (without extension). It should be the main file or one of the child files. Note that further childdoc directives such as `\childdocof` and `\childdocforward` in the indicated file will be processed in this form. The optional argument *main* passes on directly to the main file *main* while pretending to compile the child *dest*. This form behaves as if *dest* issues `\childdocof{`*main*`}` right away, and no further childdoc directives will be processed.

`\...prefix` In the alternative form `\childdocforwardprefix`,

```
\input{childdoc.def}
\childdocforwardprefix[main]{prefix}{dest}
```

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*' with *prefix* matching precisely the argument. Processing is then 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.

For example, the following file `draft.tex` with a compilation flag `\version` as described in section 2.3 compiles the main document as a draft:

```
\def\version{draft}
\input{childdoc.def}
\childdocforward{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}
\childdocforwardprefix{final}{child}
```

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

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 lines effectively perform a redirection (note that depending on the shell being used, backslashes may have to be doubled: '\' → '\\'):

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

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). The optional argument *main* can be omitted if *main* matches *dest*. 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 then forwards the processing to *main* or *dest* as described in section 2.4.

2.6 Include by Input

Including child documents by `\include` has some restrictions by design. Most notably, the content of a child document always occupies its own set of pages; pages cannot be shared between child documents. Usually, this behaviour makes perfect sense because each child document contain an essential part of the document. However, in some situations it may be desirable to compose a document from a collection of parts without having mandatory page breaks between then. For this case, the package provides a mechanism to include parts by `\input` which can also be processed individually. However, by construction this mechanism requires manual handling of the content to be output.

`\ifchilddocmanual` The main file should be prepared as usual, see section 2.1. However, the document body must make a distinction between processing of an individual part and of the main document, e.g.:

```
\ifchilddocmanual
\input{\childdocname}
\else
document body with \input{part}
\fi
```

The conditional `\ifchilddocmanual` is true whenever a part to be included by `\input` is being compiled, and the name of the part is stored in `\childdocname`.

`\childdocby` Each part to be included by `\input` should start with:

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

The directive `\childdocby` is similar to `\childdocof` described in section 2.1, but the subsequent selection of content must be done manually. To that end, both `\ifchilddoc` and `\ifchilddocmanual` will be true upon processing of a part, and the name of the part is stored in `\childdocname`. Note that `\jobname` will be set to the filename of the current part so that each part receives an individual `.aux` file that does not interfere with the `.aux` file(s) of the main document. This behaviour can be altered by the alternative form `\childdocby[*]{main}` (with a non-empty optional argument) which uses the `.aux` file of the main document by setting `\jobname` to *main*.

2.7 Driver Development

The childdoc mechanism can also be use for the development of definition files such as LATEX styles or classes. This case differs from the above setup with multiple parts included by `\include` in that no `\includeonly` should be invoked. This can be achieved by starting the include file (before `\ProvidesPackage`) with:

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

or alternatively with:

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

Both forms have slightly different effects as described above. The main file is prepared as usual, see section 2.1.

2.8 Legacy Detection

The directive `\childdocmain` in the main file can detect whether the complete document or merely a child is to be compiled even without using the directive `\childdocof`. This method is deprecated because it is less robust and there is no compelling reason to use it; it is merely provided for backward compatibility and it may be removed in future versions.

If the detection mechanism is to be used, it is mandatory to correctly specify the filename of the main file as the argument of `\childdocmain`:

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

If `\jobname` does not match the argument *main* of `\childdocmain`, it is assumed that `\jobname` points to the child file to be compiled. When using `\childdocmain` with the main file specified as argument, it suffices to start a child file with just `\input{`*main*`}` without loading of the package and using `\childdocof`. If instead all processing is done with the appropriate childdoc directives, the argument of *main* of `\childdocmain` can be empty.

An alternative version of the command line processing described in section 2.5 using the detection mechanism reads:

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

2.9 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}
```

In an alternative approach, child documents can be compiled by a specific command line without additional code or specific definitions:

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

3 Information

3.1 Copyright

This work consists of the files `README.txt`, `childdoc.ins` and `childdoc.dtx` as well as the derived files `childdoc.def`, `cdocsamp.tex` with `cdocsch1.tex`, `cdocsch2.tex`, `cdocspt3.tex`, `cdocspt4.tex`, `cdocsdrf.tex`, `cdocsfn1.tex`, `cdocsfn2.tex` as well as `childdoc.pdf`.

3.2 Files and Installation

The package consists of the files:

`README.txt`	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
`cdocspt3.tex`	sample part file
`cdocspt4.tex`	sample part 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.txt`, `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`, `cdocspt3.tex`, `cdocspt4.tex`, `cdocsdrf.tex`, `cdocsfn1.tex`, `cdocsfn2.tex`. Then copy the file `childdoc.def` to an appropriate directory of your LaTeX distribution, e.g. *texmfroot*`/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

v2.0: 2018/12/30

- immediate forward processing
- added `\childdocby` mechanism
- manual restructured

v1.6: 2018/01/17

- application for development of include files
- corrections to manual

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: 2017/04/26

- functionality in definition file

A Sample

The following presents a sample document with two chapters, two parts, a title page, a compile flag as well as three forwarding files to set the flag. It consists of eight `.tex` files:

`cdocsamp.tex`	main file
`cdocsch1.tex`	include file for chapter 1
`cdocsch2.tex`	include file for chapter 2
`cdocspt3.tex`	include file for part 3
`cdocspt4.tex`	include file for part 4
`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 eight 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{}
```

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}
```

```
\ifchilddoc
\ifchilddocmanual part\else chapter\fi:
‘\childdocname’ of ‘\childdocjob’\par
\else
main document: ‘\childdocjob’\par
\fi
version: \version\par
\end{center}
\newpage
```

Manually include selected file, otherwise process as usual:

```
\ifchilddocmanual
\section*{part ‘\childdocname’}
\input{\childdocname}
\else
```

Include the two chapters:

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

Include the two parts unless only chapters should be displayed:

```
\ifchilddoc\else
\section{part three}
\input{cdocspt3}
\section{part four}
\input{cdocspt4}
\fi
```

Process as usual until here:

```
\fi
```

End of document body:

```
\end{document}
```

Chapter Include Files. The 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
```

Part Include Files. The include files are called `cdocspt3.tex` and `cdocspt4.tex`.

Optional override for `\version` flag:

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

Include the main document:

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

Some text for part 3:

```
some text in part three
```

Some text for part 4:

```
more text in part four
```

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}
```

Command Line Processing. The following three command lines generate the output files `cdocscld`, `cdocscl1` and `cdocscl2` which should be identical to `cdocsdrf`, `cdocsch1` and `cdocsfn2`, respectively:

```
latex -jobname cdocscld \
  "\def\version{draft}\input{childdoc.def}\childdocforward{cdocsamp}"
latex -jobname cdocscl1 \
  "\input{childdoc.def}\childdocforward[cdocsamp]{cdocsch1}"
latex -jobname cdocscl2 \
  "\def\version{final}\input{childdoc.def}\childdocforward{cdocsch2}"
```

Note that the trailing backslash on each first line merely continues the input to the second line (for convenient cut ant paste). Furthermore, the command `latex` can be replaced by any of its alternative versions such as `pdflatex`.

B Implementation

This section describes 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` multiple instances have to be prevented manually:

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

`\ifchilddoc` `\ifchilddocmanual`

The conditional `\ifchilddoc` tells whether a child (true) or main (false) document is being compiled. The conditional `\ifchilddocmanual` tells whether the `\includeonly` mechanism is used (false) or the selection of child files must be performed manually (true). The definitions initialise to false:

```
\newif\ifchilddoc
\newif\ifchilddocmanual
```

`\childdocname` `\childdocjob` The macro `\childdocname` stores the name of the main document to be compiled. The macro `\childdocjob` stores the name of the document on which the LaTeX compiler was originally invoked. The content of `\jobname` cannot be compared to filenames specified in the source due to different catcodes. The following code rescans `\jobname`, stores the result in `\childdocname` and saves a copy in `\childdocjob`:

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

`\childdocdisable` The macro `\childdocdisable` prevents the main file from being processed more than once. At this stage, the main document command `\childdocmain` is assumed to be called once again where it should do nothing. Any subsequent call to it should prevent a secondary processing of the main document It overwrites the forwarding commands `\childdocof` and `\childdocforward` with empty macros to prevent further inclusions of the main document:

```
\newcommand{\childdocdisable}
{
  \renewcommand{\childdocmain}[1]{\renewcommand{\childdocmain}[1]{\endinput}}
  \renewcommand{\childdocof}[1]{}
  \renewcommand{\childdocby}[2][]{}
  \renewcommand{\childdocforward}[2][]{}
  \renewcommand{\childdocdisable}{}
}
```

`\childdocmain` The macro `\childdocmain` is to be called at the top of the main file with nothing or the main filename (without extension) as argument. First, it breaks loops. If the argument is not empty and does not match `\childdocname` (which is set by the first inclusion of `childdoc.def`), `\ifchilddoc` is set to true, `\includeonly` is applied to the child file and `\jobname` is set to the main file (for proper handling of `.aux` files):

```
\newcommand{\childdocmain}[1]
{
  \childdocdisable\childdocmain{}
  \if?#1?\else
    \begingroup
      \def\childdoctmp{#1}
      \ifx\childdoctmp\childdocname
        \def\childdoctmp{}
      \else
        \def\childdoctmp
        {
          \childdoctrue
          \includeonly{\childdocname}
          \def\childdocjob{#1}
          \def\jobname{#1}
        }
      \fi
      \expandafter
    \endgroup
    \childdoctmp
  \fi
}
```

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

```
\newcommand{\childdocof}[1]
{
  \childdocdisable
  \childdoctrue
  \includeonly{\childdocname}
  \def\jobname{#1}
  \def\childdocjob{#1}
  \input{#1}
}
```

`\childdocby` The command `\childdocby` ....

```
\newcommand{\childdocby}[2][]
{
  \childdocdisable
  \childdoctrue
  \childdocmanualtrue
  \if?#1?\else
    \def\jobname{#2}
  \fi
  \def\childdocjob{#2}
  \input{#2}
  \endinput
}
```

`\childdocforward` The command `\childdocforward` redirects compilation to the main file or (if the optional argument is given) a child file. Parameters are set as if the main file or a child file starting with `\childdocof` was compiled. Then compilation is handed over to the main file:

```
\newcommand{\childdocforward}[2][]
{
  \begingroup
    \if?#1?
      \def\childdoctmp
      {
        \def\childdocname{#2}
        \def\childdocjob{#2}
        \def\jobname{#2}
        \input{#2}
        \endinput
      }
    \else
      \def\childdoctmp
      {
        \childdocdisable
        \def\childdocname{#2}
        \childdoctrue
        \includeonly{#2}
        \def\childdocjob{#1}
        \def\jobname{#1}
        \input{#1}
        \endinput
      }
    \fi
    \expandafter
  \endgroup
  \childdoctmp
}
```

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
}
```

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

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

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

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