The extract package [*]

Hendri Adriaens
`http://stuwww.uvt.nl/~hendri`

v1.8 (2005/05/07)

Abstract

This package can be used to (conditionally) extract specific commands and environments from a source file and write them to a target file. This can be done without significant changes to the source document, but labels inside the source can be used for more flexibility. The package also provides environments to write code directly to the target file or use code in both the source and the target file. These tools allow one to generate ready-to-run files from a source document, containing only the extracted material, in an otherwise ordinary LaTeX run.

Contents

1 Introduction

I created this package when I was working on some lecture notes with exercises in the text and wanted to generate an exercises book on the fly. David Carlisle put forward the idea to use a technique as is now implemented in this package. The package heavily uses the verbatim package [13] by Rainer Schöpf and uses the xkeyval package [1] to provide a simple and easy interface[1].

There are other packages around that provide tools for conditionally typesetting material or writing material to an external file. Let me list a few.

[*]This package can be downloaded from the CTAN mirrors: `/macros/latex/contrib/extract`. See `extract.dtx` for information on installing extract into your LaTeX distribution and for the license of this package.

[1]And some tools like `\XKV@ifundefined` and `\XKV@sp@deflist`. See section 8 for more information.

askinclude [12], excludeonly [9]
: These packages enhance LaTeX's `\include` and `\includeonly` system to select the files that should be included in typesetting.

comment [6], verbatim [13], xcomment [14]
: The first two packages define the `comment` environment and the third the `xcomment` environment. These environments ignore their body. But if the command `\xcomment` is used and supplied with a list of environments, these environments will be typeset when they appear in the body of the `xcomment` environment. The comment package provides the commands `\includecomment` and `\excludecomment` to do a similar job.

optional [3], version [4], versions [8]
: These packages define some commands with which you can control which material should be typeset.

pagesel [11], pdfpages [10], selectp [2]
: These packages only typeset certain pages.

fancyvrb [15], listings [7]
: These packages (among others) provide tools to write text to an external file.

The extract package differs from all these packages since it extracts content and leaves the typeset version of the original document untouched. Furthermore, for simple extraction jobs, it is not necessary to make any changes to the document other than adding the `\usepackage` command. This allows for conditional extraction of commands and environments based on the number of the command or environment counted from the beginning of the document. More flexible conditional extraction can be achieved by adding labels to the source document. How all of this works will be explained in the sections to come.

2 Environment extraction

The following provides an example of the user interface of the package.

```
\usepackage[
  active,
  generate=file,
  extract-env={figure,table}
]{extract}
```

options
`active`
`generate`
`extract-env`

If the `active` option is not specified (or set to `false`), the package does nothing and no files are generated. If the option is specified, the package will redefine the environments `figure` and `table` so that they write their bodies (the content of the environment) to the file indicated with the `generate` option, here `file.tex`[2], including the `\begin{figure}` and `\end{figure}` commands. Besides that, the environments will be executed as usual. Most environments are supported (known exception is the `document` environment). When the package encounters `\begin{document}` or `\end{document}` in the source file, by default[3], it will also write these commands to the target file, such that, if a suitable preamble is added, the file is ready to be run by

[2] If a file extension is lacking, `.tex` will be used.
[3] See section 5.3 for the `handles` option.

LaTeX. See listing 1 for an example.[4] The package will also write a header to the target

xtrex1.tex

```
\documentclass[10pt]{article}
\usepackage[
  active,
  generate=file,
  copydocumentclass=false,
  extract-env=equation
]{extract}
\begin{extract}
\documentclass[11pt]{article}
\end{extract}
\begin{document}
Some text.
\begin{equation}
a^2+b^2=c^2
\end{equation}
Some text.
\begin{equation}
x^2+y^2=z^2
\end{equation}
Some text.
\end{document}
```

file.tex

```
\documentclass[11pt]{article}

\begin{document}

\begin{equation}
a^2+b^2=c^2
\end{equation}

\begin{equation}
x^2+y^2=z^2
\end{equation}

\end{document}
```

Listing 1: Environment extraction.

file with some information about the source and time of generation of the target file. This header can be turned off with the `no-header` option. See also section 5.3.

3 Command extraction

options
`extract-cmd`
`extract-cmdline`

This package can also extract commands. It supplies two methods to do this. See the example below.

```
\usepackage[
  active,
  generate=file,
  extract-cmd=section,
  extract-cmdline=label
]{extract}
```

The first method (accessed with the `extract-cmd` option) is based on the particular syntax of the command and hence only supports particular commands. It will read it arguments and write them, together with the original command, to the target file. Besides that, the command will be executed as usual. Currently, the commands `\chapter`, `\section`, `\subsection`, and `\subsubsection` in the standard LaTeX classes (and classes or packages derived from those) are supported. An optional argument to these commands, like `\chapter[short]{long}` is supported. However, the starred version `\chapter*{title}` will not be written to the file due to technical limitations. Sections 6 and 8 will explain this in more detail.

The second method (accessed with the `extract-cmdline` option) will redefine commands to write themselves and all the text following on the same line to the target

[4]See section 5.1 for the `extract` environment, section 5.3 for the `copydocumentclass` option and section 7 how to obtain the example files.

file and will also execute the entire line as with an ordinary LaTeX run. This allows to redefine any command that is not supported by the first method, but should be applied with care. If the command is used internally in a class or style file, your document might fail to run. In particular, one should not redefine one of the commands supported by the first method with this method. See listing 2 for a working example of both methods.[5]

`xtrex2.tex`

```
\documentclass{article}
\usepackage[
  active,
  generate=file,
  extract-env=exercise,
  extract-cmd=section,
  extract-cmdline=label
]{extract}
\begin{extract*}
\newtheorem{exercise}{Exercise}
\end{extract*}
\begin{document}
\section{Theory}
\label{sec:1}
Some text.
\section{Exercises}
\begin{exercise}
Use the results from section
\ref{sec:1} to show that\dots
\end{exercise}
Some text.
\end{document}
```

`file.tex`

```
\documentclass{article}
\newtheorem{exercise}{Exercise}

\begin{document}

\section{Theory}

\label{sec:1}

\section{Exercises}

\begin{exercise}
Use the results from section
\ref{sec:1} to show that\dots
\end{exercise}

\end{document}
```

Listing 2: Command extraction.

4 Conditional extraction

4.1 Extraction with numbers

options `-nrs` It is also possible to conditionally extract environments and commands. After the options `extract-env`, `extract-cmd` and `exstract-cmdline` are used, for each environment or command specified there, there will be a new option with the name of that environment or command and the `-nrs` postfix. This option can take a comma separated list in which you can specify which environments or macros (counting from the `\usepackage{extract}` command[6]) should be extracted. Table 1 on page 5 lists the syntax that can be used in the comma separated list. The term ‘item’ is used for ‘command or environment’. See an example in the listing below.

```
\documentclass{book}
\usepackage[
  active,
  generate=file,
```

[5]See section 5.1 for the `extract*` environment.

[6]Notice that starred commands like `\chapter*`, which won’t be redefined by the `extract-cmd` option, will also not be counted.

Syntax	Meaning
⟨*x*⟩	Item ⟨*x*⟩ will be extracted.
-⟨*x*⟩	All items up to and including item ⟨*x*⟩ will be extracted.
⟨*x*⟩-	All items from and including item ⟨*x*⟩ will be extracted.
⟨*x*⟩-⟨*y*⟩	All items in between ⟨*x*⟩ and ⟨*y*⟩, including ⟨*x*⟩ and ⟨*y*⟩ will be extracted.

Table 1: Syntax for conditional extraction with numbers.

```
  extract-env={figure,table},
  figure-nrs={-2,4-},
  extract-cmd=chapter,
  chapter-nrs={3-5,7}
]{extract}
\begin{document}
...
\end{document}
```

This example, when completed with content, will extract all `table` environments, `figure` environments 1, 2, and all figures from (and including) figure 4. It will also extract chapter commands 3 to 5 and 7.

4.2 Extraction with labels

options `-labels`

Conditional extraction is also possible with labels. The advantage of using labels is that output does not change (in comparison to using numbers) when commands or environments are added. The drawback is that one needs to modify the source document and add labels in the text.

`\extractionlabel`

Labels are declared with the following command.

```
\extractionlabel{⟨name⟩}
```

A label should be declared just before the command or environment that you want to extract. For instance

```
\extractionlabel{exer-a}
\begin{exercise}
...
\end{exercise}
```

You can reuse the same label multiple times and you can specify which items should be extracted by the options with a `-labels` prefix. This works in the same way as with numbers.

```
\documentclass{book}
\usepackage[
  active,
  generate=file,
  extract-env=exercise,
  exercise-labels={exer-a,exer-c}
]{extract}
\begin{document}
...
\end{document}
```

This example will only extract exercises that have been preceded by the declaration `\extractionlabel{exer-a}` or `\extractionlabel{exer-c}`.

When using both conditional extraction with numbers and with labels, the command or environment at hand will be extracted when at least on of the conditions is true. Find an example in listing 3.

xtrex3.tex

```
\documentclass{article}
\usepackage[
  active,
  generate=file,
  extract-env=figure,
  figure-nrs={1,3},
  figure-labels={fig-a,fig-b}
]{extract}
\begin{document}
Some text.
\begin{figure}
Figure 1.
\end{figure}
Some text.
\extractionlabel{fig-a}
\begin{figure}
Figure 2.
\end{figure}
Some text.
\extractionlabel{fig-b}
\begin{figure}
Figure 3.
\end{figure}
Some text.
\extractionlabel{fig-c}
\begin{figure}
Figure 4.
\end{figure}
\end{document}
```

file.tex

```
\documentclass{article}

\begin{document}

\begin{figure}
Figure 1.
\end{figure}

\begin{figure}
Figure 2.
\end{figure}

\begin{figure}
Figure 3.
\end{figure}

\end{document}
```

Listing 3: Conditional extraction.

5 Extraction tools

5.1 Manual extraction

environment `extract`

The package provides the environment `extract`.

```
\begin{extract}
⟨body⟩
\end{extract}
```

This environment writes its body only to the target file. This can be used to generate a preamble in the target file so that you can run the generated file through LaTeX immediately after creation. See the example in listing 1 on page 3 and the example below.

environment `extract*`

There also exists a starred version of the `extract` environment.

```
\begin{extract*}
```

```
⟨body⟩
\end{extract*}
```

This environment will not only write the body to the target file, but will also execute the code in the source document. This can be used to create a common preamble which holds packages and commands that will be used in both the source and the target file. See listing 2 on page 4 and listing 4 for examples. Notice that the 'answer' will not appear in `xtrex4.dvi`.

`xtrex4.tex`

```
\documentclass{article}
\usepackage[
  active,
  generate=file,
  extract-env=equation*
]{extract}
\begin{extract*}
\usepackage{amsmath}
\end{extract*}
\begin{document}
Some text.
\begin{equation*}
x^2+y^2=z^2
\end{equation*}
\begin{extract}
$x=3$, $y=4$ and $z=5$
satisfy this equation.
\end{extract}
\end{document}
```

`file.tex`

```
\documentclass{article}
\usepackage{amsmath}

\begin{document}

\begin{equation*}
x^2+y^2=z^2
\end{equation*}
$x=3$, $y=4$ and $z=5$
satisfy this equation.

\end{document}
```

Listing 4: Extract environments.

`\extractline`
`\extractline*`

The package also provides a command that extracts the current line.

```
\extractline
\extractline*
```

The starred version also executes the code at that line. See the example below.

```
\extractline This line should be extracted.
\extractline*This line should be extracted and executed.
```

Notice that a space is following the command `\extractline` and that no space is following the command `\extractline*`. In the first line, LaTeX will eat this space, but it won't do that in the second line. If we add a space there in between the `*` and `This`, this space will be executed and extracted as well.

`\extractline`
`\extractline*`
environments
`extract`
`extract*`

The `extract` and `extract*` environments and the macros `\extractionlabel` and `\extractionlabel*` have an optional argument for specifying the label directly.

```
\begin{extract}[⟨name⟩]
\begin{extract*}[⟨name⟩]
\extractline[⟨name⟩]
\extractline*[⟨name⟩]
```

options
extract-nrs
line-nrs
extract-labels
line-labels

The options extract-nrs and line-nrs can be used to control conditional extraction of these environments and commands using numbers. Moreover, one can do conditional extractions with labels for these commands and environments as well. Use the options extract-labels and line-labels for that purpose. See also section 4 for information about conditional extraction.

See an example in listing 5. This example will demonstrate that only certain lines and environments are extracted. Note that, when running xtrex5.tex with LaTeX, the output, xtrex5.dvi, will contain the following.

```
line 2
line 4
line 5
```

xtrex5.tex

```
\documentclass{article}
\usepackage[
  active,
  generate=file,
  copydocumentclass=false,
  extract-labels=type-a,
  line-labels={type-a,type-c},
  line-nrs=3
]{extract}
\begin{extract}[type-a]
\documentclass{article}
\end{extract}
\begin{extract}[type-b]
\documentclass{book}
\end{extract}
\begin{document}
\parindent0pt
\extractline[type-a]line 1\\
\extractline*line 2\\
\extractline line 3\\
\extractline*[type-a]line 4\\
\extractline*[type-c]line 5\\
\extractline line 6\\
\end{document}
```

file.tex

```
\documentclass{article}

\begin{document}
line 1\\
line 3\\
line 4\\
line 5\\

\end{document}
```

Listing 5: Optional labels.

5.2 Skipping extraction

environment
extractskip

The package also provides the extractskip environment.

```
\begin{extractskip}[⟨name⟩]
⟨body⟩
\end{extractskip}
```

The body of this environment will not be written to the target file, but will be executed. This environment can be used to skip material in an environment which extracts its entire body. This could be the extract environment, but also an environment that has been redefined to be extracted using the extract-env option. The argument ⟨*name*⟩ is optional and contains the label.

options `extractskip-nrs` `extractskip-labels`

This environment can also operate conditionally as has been described in section 4 using the options `extractskip-nrs` and `extractskip-labels`. Listings 6 and 7 on page 10 will demonstrate this environment.

`xtrex6.tex`

```
\documentclass{article}
\usepackage[
  active,
  generate=file,
  extract-env=figure
]{extract}
\begin{document}
\begin{figure}[!h]
\begin{extractskip}
\fbox{figure 1}
\end{extractskip}
\fbox{figure 2}
\end{figure}
\begin{extract*}
\begin{itemize}
\item 1
\item 2
a\begin{extractskip}b
\item 3
c\end{extractskip}d
\item 4
\begin{extractskip}
\item 5
\end{extractskip}
\end{itemize}
\end{extract*}
\end{document}
```

`file.tex`

```
\documentclass{article}

\begin{document}

\begin{figure}[!h]
\fbox{figure 2}
\end{figure}
\begin{itemize}
\item 1
\item 2
a
d
\item 4
\end{itemize}

\end{document}
```

Listing 6: Skipping extraction.

5.3 Miscellaneous options

option `header`

When setting the `header` option to `false`, no header will be written to the target file. If the option is not specified or set to `true`, the package will write a header to the target file including information on when the target file was generated and which source file was used.

option `handles`

This option controls whether the package will write `\begin{document}` and `\end{document}` to the target file when it encounters these commands in the source document. By default, this option is set to `true`. When the option is set to `false`, the generated file can be `\input`ed or `\include`d by another file immediately after production.

option `copydocumentclass`

This option control whether the target file should get the same `\documentclass` command as the source document (including class options). If set to `false`, you should specify another document class for the target file, for instance using the `extract` environment. See listing 1 on page 3 for an example.

xtrex7.tex

```
\documentclass{article}
\usepackage[
  active,
  generate=file,
  extractskip-labels=skipb
]{extract}
\begin{document}
\begin{extract*}
\begin{itemize}
\begin{extractskip}[skipa]
\item 1
\end{extractskip}
\begin{extractskip}[skipb]
\item 2
\end{extractskip}
\begin{extractskip}[skipc]
\item 3
\end{extractskip}
\end{itemize}
\end{extract*}
\end{document}
```

file.tex

```
\documentclass{article}

\begin{document}
\begin{itemize}
\item 1
\item 3
\end{itemize}

\end{document}
```

Listing 7: Skipping extraction conditionally.

6 How it works, limitations

The package works as follows. When an environment is asked to be extracted, the package will first make a backup of the environment with the `XTR` prefix, for instance, `XTRequation*`. After that, the package will redefine the environment to read the lines of its body verbatim (without executing), parse the lines to locate `extractskip` environments and write all lines to a temporary file and the lines that are not in an `extractskip` environment to a temporary file. After the environment is finished, the temporary file, containing for instance

```
\begin{XTRequation*}
x^2+y^2=z^2
\end{XTRequation*}
```

will be inserted in the source document using `\input`. This works, at least in theory, with most environments. Notice that this method does require a change to the `\begin` and `\end` macros provided by the LaTeX kernel [5]. I had to add a hook to these commands to be able to collect code to be executed after the current environment is ended. In particular, this package will use those hooks to `\input` the original code after finishing the current group. See section 8 for more details.

Commands require a different approach. As a command can have a very specific argument structure, redefining commands safely, without distorting its original behavior, is not possible in general. I have chosen to support the most basic document structure commands as provided by standard LaTeX classes, like book and article. Extraction of commands from other classes, like provided by the koma-script bundle, might work, but there is no guarantee.

To be more precise: when requested, the macros `\@chapter` and `\@sect` will be redefined. These commands are at the basis of all chapter and section commands. When extract redefines one of these commands, it first makes a backup, like `\XTR@chapter`.

After that, it will redefine the original command to read its argument(s), export them to the target file and execute the backup with the proper arguments.

An alternative to this, rather restricted method is the method accessed by the `extract-cmdline` option which redefines the commands listed there to read the entire line of text, write that to the target file and afterwards execute it with a backup copy of the original command. This methods too has its drawbacks though as you can't redefine commands that are used internally in other macros.

More details on the package code can be found in section 8.

7 Source and examples

To generate this documentation, find the source of this package, `extract.dtx` in your local LaTeX installation or on CTAN and perform the following steps.

```
latex extract.dtx
latex extract.dtx
bibtex extract
makeindex -s gglo.ist -o extract.gls extract.glo
makeindex -s gind.ist -o extract.ind extract.idx
latex extract.dtx
latex extract.dtx
```

If you only want to produce the package and example files from the source, then the first step is sufficient. This step will generate the package file `extract.sty` and the example files `xtrex1.tex`, `xtrex2.tex`, `xtrex3.tex`, `xtrex4.tex`, `xtrex5.tex`, `xtrex6.tex` and `xtrex7.tex`.

8 Implementation

Initializations.

```
%<*extract>
\NeedsTeXFormat{LaTeX2e}[1995/12/01]
\ProvidesPackage{extract}
  [2005/05/07 v1.8 extract content from document (HA)]
\RequirePackage{verbatim}
\RequirePackage{xkeyval}
\newwrite\XTR@out
\newwrite\XTR@tmp
\newif\ifXTR@st
\newif\ifXTR@skip
\newif\ifXTR@extract
```

`\XTR@err` {⟨*text*⟩}

Error macro.

```
\def\XTR@err#1{\PackageError{extract}{#1}\@ehc}
```

`\XTR@namelet` {⟨*cmd1*⟩}{⟨*cmd2*⟩}

Version of `\let` for two command sequence names.

```
\def\XTR@namelet#1#2{%
  \expandafter\let\csname#1\expandafter\endcsname\csname#2\endcsname
}
```

option active

Options section, powered by xkeyval. Control extraction with one switch.

```
\define@boolkey[XTR]{extract.sty}[XTR@]{active}[true]{}
```

option header

Do not create a header in the target file.

```
\define@boolkey[XTR]{extract.sty}[XTR@]{header}[true]{}
```

option handles

Extract `\begin{document}` and `\end{document}` or not.

```
\define@boolkey[XTR]{extract.sty}[XTR@]{handles}[true]{}
```

option copydocumentclass

Copy the `\documentclass` command to the target file.

```
\define@boolkey[XTR]{extract.sty}[XTR@]{copydocumentclass}[true]{}
```

option generate

Entry point for the target file name.

```
\DeclareOptionX[XTR]{generate}{\lowercase{\def\XTR@file{#1}}}
```

option extract-env

Environments that should be extracted.

```
\DeclareOptionX[XTR]{extract-env}{%
  \def\XTR@envs{#1}%
  \XKV@for@n{#1}\XTR@tempa\XTR@tempb
}
```

option extract-cmd

Commands that should be extracted with the 'arguments method'.

```
\DeclareOptionX[XTR]{extract-cmd}{%
  \def\XTR@cmdsargs{#1}%
  \XKV@for@n{#1}\XTR@tempa\XTR@tempb
}
```

option extract-cmdline

Commands that should be extracted with the 'line method'.

```
\DeclareOptionX[XTR]{extract-cmdline}{%
  \def\XTR@cmdsline{#1}%
  \XKV@for@n{#1}\XTR@tempa\XTR@tempb
}
\def\XTR@tempb{%
```

options -nrs -labels

For each environment or command provide new package options that save the argument to a list cleared from redundant spaces. The `-nrs` options also create a 'counter' for counting the commands or environments. The lists and counters will be used for conditional extraction. Note that `\XTR@tkey` contains the key name inside the option macro (due to the use of `\ProcessOptionsXi`).

```
  \DeclareOptionX[XTR]{\XTR@tempa-nrs}{%
    \expandafter\XKV@sp@deflist\csname XTR@\XKV@tkey\endcsname{##1}%
    \XTR@namelet{XTR@\XKV@tkey @cnt}{z@}%
  }%
  \DeclareOptionX[XTR]{\XTR@tempa-labels}{%
    \expandafter\XKV@sp@deflist\csname XTR@\XKV@tkey\endcsname{##1}%
  }%
}
```

options line-nrs line-labels

Generate options for `\extractline` commands.

```
\def\XTR@tempa{line}\XTR@tempb
```

options extract-nrs extract-labels

Generate options for `extract` environments.

```
\def\XTR@tempa{extract}\XTR@tempb
```

options extractskip-nrs extractskip-labels

Generate options for `extractskip` environments.

```
\def\XTR@tempa{extractskip}\XTR@tempb
```

Generate an error for unknown options.

```
\DeclareOptionX*{\XTR@err{Unknown option `\CurrentOption'}}
```

Initialize options.

```
\ExecuteOptionsX[XTR]{header=true,handles=true,copydocumentclass=true}
```

Process options.

```
\ProcessOptionsX[XTR]
```

\XTR@opentmp \XTR@writetmp \XTR@closetmp \XTR@writeout

Shortcut macros for much used command sequences.

```
\def\XTR@opentmp{\immediate\openout\XTR@tmp\jobname.xtr\relax}
\def\XTR@writetmp{\immediate\write\XTR@tmp}
\def\XTR@closetmp{\immediate\closeout\XTR@tmp}
\def\XTR@writeout{\immediate\write\XTR@out}
```

Perform some checks on the input. Notice the use of `\XKV@ifundefined` which is equal to `\@ifundefined` if no ε-TEX engine is available and which uses `\ifcsname` when it is. In the latter case, testing whether commands are defined does not create an entry in TEX's hash table.

```
\ifXTR@active
  \XKV@ifundefined{XTR@file}{
    \XTR@activefalse
    \XTR@err{no file to generate; extract deactivated}
  }{}
  \XTR@opentmp
  \XTR@writetmp{%
    \string\lowercase{\string\def\string\XTR@tempa{\jobname}}%
  }
  \XTR@closetmp
  \input{\jobname.xtr}
  \ifx\XTR@tempa\XTR@file
    \XTR@activefalse
    \XTR@err{attempt to overwrite source file; extract deactivated}
  \fi
\fi
```

\@envdepth

Counter for depth of environments.

```
\newcount\@envdepth\@envdepth\z@
```

\begin {⟨*environment*⟩}

Modify the macro `\begin` to allow adding code to a level specific hook which can be executed after `\endgroup` in `\end`. See for more info on this macro the LaTeX source [5].

```
\def\begin#1{%
  \@ifundefined{#1}%
    {\def\reserved@a{\@latex@error{Environment #1 undefined}\@eha}}%
    {\def\reserved@a{\def\@currenvir{#1}%
     \edef\@currenvline{\on@line}%
     \csname #1\endcsname}}%
  \@ignorefalse
  \begingroup\@endpefalse
```

Advance depth level.

```
  \global\advance\@envdepth\@ne
```

Initialize the hook for this level.

```
  \global\@namedef{@afterendenvhook@\romannumeral\@envdepth}{}%
  \reserved@a}
```

`\end` {⟨*environment*⟩}
Modify `\end` to execute the code collected in the hook.

```
\def\end#1{%
  \csname end#1\endcsname\@checkend{#1}%
  \expandafter\endgroup\if@endpe\@doendpe\fi
```

Copy current hook code to a temporary macro.

```
  \expandafter\let\expandafter\reserved@a
    \csname @afterendenvhook@\romannumeral\@envdepth\endcsname
```

Decrease the depth.

```
  \global\advance\@envdepth\m@ne
```

Execute the hook of the current environment. This is done after decreasing the depth as to avoid level mixing problems when the hook contains another environment. This environment has to be executed at the same level as the environment in which the hook was defined since it is executed after the group and does not below anymore to the environment in which the hook was defined.

```
  \reserved@a\relax
  \if@ignore\@ignorefalse\ignorespaces\fi}
```

`\AfterEndEnv` Adds code to the macros `\@afterendenvhook@i`, `ii`, etc. which will be executed after the group of the current environment.

```
\def\AfterEndEnv{%
  \expandafter\g@addto@macro
    \csname @afterendenvhook@\romannumeral\@envdepth\endcsname
}
```

`\XTR@checkxtr` {⟨*type*⟩}{⟨*item*⟩}
Checks whether a certain environment or command should be extracted or skipped. ⟨*type*⟩ is the type of check: for extraction or for skipping content. ⟨*item*⟩ is the name of a command or an environment.

```
\def\XTR@checkxtr#1#2{%
  \@nameuse{XTR@#1false}%
  \XTR@namelet{XTR@maketrue}{XTR@#1true}%
```

First check whether, for this macro or environment, some method of conditional extraction is used. If not, just extract.

```
  \XKV@ifundefined{XTR@#2-nrs}{%
    \XKV@ifundefined{XTR@#2-labels}\XTR@maketrue{}%
  }{%
```

Advance the 'counter'.

```
    \begingroup
      \expandafter\count@\csname XTR@#2-nrs@cnt\endcsname
      \advance\count@\@ne
      \edef\XTR@resa{\expandafter\noexpand\expandafter\gdef\expandafter
        \noexpand\csname XTR@#2-nrs@cnt\endcsname{\the\count@}}%
    \expandafter\endgroup\XTR@resa
  }%
  \@nameuse{ifXTR@#1}\else
    \XKV@ifundefined{XTR@#2-labels}{}{%
```

If the current label is in the list for extraction, extract it.

```
      \ifx\XTR@currentlabel\relax\else
```

```
        \@expandtwoargs\in@{,\XTR@currentlabel,}%
          {,\csname XTR@#2-labels\endcsname,}%
        \ifin@\XTR@maketrue\fi
      \fi
    }%
  \fi
  \@nameuse{ifXTR@#1}\else
    \XKV@ifundefined{XTR@#2-nrs}{}{%
```

If the current command or environment number is in the list, extract it.

```
      \expandafter\XTR@ch@ckxtr\csname XTR@#2-nrs\expandafter
        \endcsname\csname XTR@#2-nrs@cnt\endcsname
    }%
  \fi
```

Redefine `\XTR@currentlabel` to avoid extracting all following environments of this type.

```
  \global\let\XTR@currentlabel\relax
}
```

`\XTR@ch@ckxtr` ⟨*list*⟩⟨*counter*⟩

Parse the ⟨*list*⟩ of numbers and compare each item with the ⟨*counter*⟩ holding the number of the current item.

```
\def\XTR@ch@ckxtr#1#2{%
  \XKV@for@o#1\XTR@resa{\expandafter\XTR@ch@ck@tr\XTR@resa--\@nil#2}%
}
```

`\XTR@ch@ck@tr` ⟨*x*⟩`-`⟨*y*⟩`-`⟨*z*⟩`\@nil`⟨*counter*⟩

Parse an element of the list. Basically, decide whether we have `x`, `x-y`, `x-` or `-y` and act accordingly.

```
\def\XTR@ch@ck@tr#1-#2-#3\@nil#4{%
  \ifx\@empty#1\@empty
    \ifnum#4>#2 \else\XTR@maketrue\fi
  \else
    \ifx\@empty#2\@empty
      \ifx\@empty#3\@empty
        \ifnum#4=#1 \XTR@maketrue\fi
      \else
        \ifnum#4<#1 \else\XTR@maketrue\fi
      \fi
    \else
      \ifnum#4<#1 \else\ifnum#4>#2 \else\XTR@maketrue\fi\fi
    \fi
  \fi
}
```

`\extractionlabel` `\XTR@currentlabel`

`\extractionlabel` saves its argument (after removing redundant spaces). This label will be used for conditional extraction. `\XTR@currentlabel` is initialized.

```
\def\extractionlabel{\KV@@sp@def\XTR@currentlabel}
\let\XTR@currentlabel\relax
```

`\extract` `\extract*`

Define environments that write verbatim to the target file. The starred version also executes the code by writing it to a temp file and inputting it `\AfterEndEnv`, just as with redefining existing environments. When the package is inactive, `extract` is equivalent to the `comment` environment and `extract*` takes its body out of the group and executes it hence acting as if `\begin{extract*}` and `\end{extract*}` were never typed.

```
\def\extract{\XTR@stfalse\XTR@extract}
\@namedef{extract*}{\XTR@sttrue\XTR@extract}
```

`\XTR@extract` Prepare verbatim reading and check for an optional argument.

```
\def\XTR@extract{%
  \@bsphack
  \let\do\@makeother\dospecials\catcode`\^^M\active
  \@testopt\XTR@@xtract\@nil
}
```

`\XTR@@xtract` `[`⟨*label*⟩`]`

Process the optional label, define line processing and start reading verbatim. Do not extract when the package is not active. Use a temporary file to extract the body to in case this needs to be executed in the source document (`extract*` environment).

```
\def\XTR@@xtract[#1]{%
```

Check state.

```
  \ifXTR@active
    \def\XTR@tempa{#1}%
    \ifx\XTR@tempa\@nnil\else
      \KV@@sp@def\XTR@currentlabel{#1}%
    \fi
    \XTR@checkxtr{extract}{extract}%
  \else
    \XTR@extractfalse
  \fi
  \ifXTR@st\XTR@opentmp\fi
  \let\verbatim@processline\XTR@processline@begin
  \verbatim@start
}
```

```
\begingroup
  \lccode`\!=`\\ \lccode`\(=`\{ \lccode`\)=`\}
\lowercase{\endgroup
```

`\XTR@processline@begin` This macro starts the reparsing of a line read by verbatim. It is possible to have `\begin{extractskip}` and `\end{extractskip}` on the same line.

```
\def\XTR@processline@begin{%
```

Initialize `\@temptokena` (used for temp file) and `\verbatim@line` (used for output file). Save the original content of `\verbatim@line` for later use.

```
  \@temptokena{}%
  \edef\XTR@orig@line{\the\verbatim@line}%
  \verbatim@line{}%
  \expandafter\XTR@testbegin\XTR@orig@line!begin(extractskip)\@nil
}
```

`\XTR@testbegin` ⟨*text1*⟩`\begin{extractskip}`⟨*text2*⟩`\@nil`

Checks whether `\begin{extractskip}` occurs.

```
\def\XTR@testbegin#1!begin(extractskip)#2\@nil{%
  \@temptokena\expandafter{\the\@temptokena#1}%
  \verbatim@line\expandafter{\the\verbatim@line#1}%
  \def\XTR@tempa{#2}%
```

If ⟨*text2*⟩ empty, there is no `\begin{extractskip}`. Just write the content to file.

```
  \ifx\XTR@tempa\@empty\XTR@processline@write\else\XKV@afterfi
```

Check the label.

```
    \XTR@skiplabel#2[]\@nil
```

Should we skip the `extractskip` environment or not?

```
    \XTR@checkxtr{skip}{extractskip}%
```

Switch to scanning for `\end{extractskip}` in the next line.

```
    \let\verbatim@processline\XTR@processline@end
```

Remove some stuff that we added and continue scanning for `\end{extractskip}` on the current line.

```
    \ifx\XTR@tempa\@nnil\XKV@afterelsefi
      \XTR@t@stbegin#2\@nil
    \else\XKV@afterfi
      \expandafter\XTR@t@stbegin\XTR@tempa\@nil
    \fi
  \fi
}
```

`\XTR@skiplabel` ⟨*text1*⟩`[`⟨*label*⟩`]`⟨*text2*⟩`\@nil`

This macro checks whether a label is present and sets `\XTR@currentlabel` if necessary.

```
\def\XTR@skiplabel#1[#2]#3\@nil{%
  \def\XTR@tempa{#1}%
  \def\XTR@tempb{#2}%
  \ifx\XTR@tempa\@empty
    \ifx\XTR@tempb\@empty
      \let\XTR@tempa\@nnil
    \else
      \KV@@sp@def\XTR@currentlabel{#2}%
      \XTR@sk@plabel#3\@nil
    \fi
  \else
    \let\XTR@tempa\@nnil
  \fi
}
```

`\XTR@sk@plabel` ⟨*text*⟩`[]\@nil`

Remove extra brackets from input.

```
\def\XTR@sk@plabel#1[]\@nil{\def\XTR@tempa{#1}}
```

`\XTR@t@stbegin` ⟨*text*⟩`\begin{extractskip}\@nil`

Remove the extra `\begin{extractskip}` and start scanning for `\end{extractskip}` in the current line.

```
\def\XTR@t@stbegin#1!begin(extractskip)\@nil{%
  \XTR@testend#1!end(extractskip)\@nil
}
```

\XTR@processline@end Starts scanning for \end{extractskip} in case this was not on one line together with \begin{extractskip}. Comparable to \XTR@processline@begin.

```
\def\XTR@processline@end{%
  \@temptokena{}%
  \edef\XTR@orig@line{\the\verbatim@line}%
  \verbatim@line{}%
  \expandafter\XTR@testend\XTR@orig@line!end(extractskip)\@nil
}
```

\XTR@testend ⟨*text1*⟩\end{extractskip}⟨*text2*⟩\@nil
Check whether \end{extractskip} occurs in the line.

```
\def\XTR@testend#1!end(extractskip)#2\@nil{%
  \@temptokena\expandafter{\the\@temptokena#1}%
```

Skip material conditionally on labels or numbers.

```
  \ifXTR@skip\else\verbatim@line\expandafter{\the\verbatim@line#1}\fi
  \def\XTR@tempa{#2}%
  \ifx\XTR@tempa\@empty\XTR@processline@write\else\XKV@afterfi
```

Switch to scanning for \begin{extractskip} in the next line.

```
    \let\verbatim@processline\XTR@processline@begin
```

Continue scanning for \begin{extractskip} in the current line.

```
    \XTR@t@stend#2\@nil
  \fi
}
```

\XTR@t@stend ⟨*text*⟩\end{extractskip}\@nil
Remove the redundant \end{extractskip} and continue scanning for for the string \begin{extractskip} in this line.

```
\def\XTR@t@stend#1!end(extractskip)\@nil{%
  \XTR@testbegin#1!begin(extractskip)\@nil
}}
```

\XTR@processline@write Writes the material to the appropriate file. If one of the tokens has become empty, it might be because the line was empty originally or because the parsing and removal of \begin{extractskip} and \end{extractskip} made it empty. In the latter case, do not write the empty line. In the former case, do write it.

```
\def\XTR@processline@write{%
  \ifXTR@st\ifcat$\the\@temptokena$\else
    \XTR@writetmp{\the\@temptokena}%
  \fi\fi
  \ifXTR@extract\ifcat$\the\verbatim@line$\else
    \XTR@writeout{\the\verbatim@line}%
  \fi\fi
  \ifx\XTR@orig@line\@empty\XTR@writetmp{}\XTR@writeout{}\fi
}
```

\endextract
\endextract* Stop reading verbatim and if necessary execute the body of the environment after the extract* environment.

```
\def\endextract{\XTR@stfalse\XTR@endextract}
\@namedef{endextract*}{\XTR@sttrue\XTR@endextract}
\def\XTR@endextract{%
  \@esphack
  \ifXTR@st
    \XTR@closetmp
    \AfterEndEnv{\input{\jobname.xtr}}%
  \fi
}
```

\extractskip \endextractskip The `extractskip` environment when it is not used inside an environment that is redefined to be extracted. Hence this environment makes itself disappear, just as `extract*`, but doesn't write to the output file. The trick with `XTR@activefalse` will remain local.

```
\@namedef{extractskip}{\XTR@activefalse\@nameuse{extract*}}
\XTR@namelet{endextractskip}{endextract*}
```

\extractline This macro extracts all text after the macro and at the same line. First we check for an optional star.

```
\def\extractline{%
  \XKV@ifstar{\XTR@sttrue\XTR@extractline}%
    {\XTR@stfalse\XTR@extractline}%
}
```

\XTR@extractline Start the group and reset all catcodes for verbatim reading.

```
\def\XTR@extractline{%
  \begingroup
    \let\do\@makeother\dospecials\catcode`\^^M\active
```

Test for an optional argument. Note that, due to reset catcodes, macros won't work in the optional argument, but that is not a real restriction, while it saves some tokens and memory. If we want to allow for macro arguments, we need an extra macro for the check.

```
    \@testopt\XTR@@xtractline\@nil
}
```

\XTR@@xtractline `[`⟨*label*⟩`]`⟨*text*⟩⟨*eol*⟩
The workhorse that reads input until the end of the line. Use the `\lowercase` trick for the definition.

```
\begingroup
  \catcode`\~=\active\lccode`\~=`\^^M
\lowercase{\endgroup
  \def\XTR@@xtractline[#1]#2~{%
```

Check state.

```
      \ifXTR@active
        \def\XTR@tempa{#1}%
        \ifx\XTR@tempa\@nnil\else
          \KV@@sp@def\XTR@currentlabel{#1}%
        \fi
        \XTR@checkxtr{extract}{line}%
      \else
        \XTR@extractfalse
      \fi
      \ifXTR@extract\XTR@writeout{#2}\fi
```

If we need to execute the line, the catcodes are wrong, so write it to the temporary file and insert it again when the catcodes are reset by `\endgroup`.

```
      \ifXTR@st\XTR@opentmp\XTR@writetmp{#2}\XTR@closetmp\fi
    \endgroup
```

Insert original content.

```
    \ifXTR@st
      \input{\jobname.xtr}%
    \fi
  }%
}
```

Only define the following macros when the package is active. This branch also performs redefinitions of the macros and environments that should be extracted.

```
\ifXTR@active
```

Start writing the target file.

```
\immediate\openout\XTR@out\XTR@file\relax
```

Write header to the target file.

```
\ifXTR@header
```

Compute the time.

```
  \@tempcnta\time
  \divide\@tempcnta 60
  \edef\XTR@tempb{%
    \the\year/\ifnum\the\month<10 0\fi\the\month/%
    \ifnum\the\day<10 0\fi\the\day,\the\@tempcnta:%
  }
  \multiply\@tempcnta 60
  \@tempcntb\time
  \advance\@tempcntb-\@tempcnta
  \ifnum\@tempcntb<10
    \xdef\XTR@tempb{\XTR@tempb0\the\@tempcntb}
  \else
    \xdef\XTR@tempb{\XTR@tempb\the\@tempcntb}
  \fi
  \begingroup
```

Save the % character.

```
    \catcode`\%=12
    \gdef\XTR@tempa{%%\space}
  \endgroup
```

Write all information to the target file.

```
  \XTR@writeout{\XTR@tempa}
  \filename@parse\XTR@file
  \ifx\filename@ext\relax\def\filename@ext{tex}\fi
  \XTR@writeout{%
    \XTR@tempa This is file, `\filename@base.\filename@ext',%
  }
  \XTR@writeout{%
    \XTR@tempa generated with the extract package.^^J\XTR@tempa
  }
  \XTR@writeout{\XTR@tempa Generated on : \space\XTR@tempb}
  \filename@parse\jobname
  \ifx\filename@ext\relax\def\filename@ext{tex}\fi
  \XTR@writeout{%
```

```
    \XTR@tempa From source \space: \space\filename@base.\filename@ext
  }
  \XTR@writeout{%
    \XTR@tempa Using options: \space\csname opt@extract.sty\endcsname
  }
  \XTR@writeout{\XTR@tempa}
\fi
```

If requested, reconstruct the \documentclass command using information from xkeyval.

```
\ifXTR@copydocumentclass
  \def\XTR@tempa#1.cls\@nil{\def\XTR@tempa{#1}}
  \expandafter\XTR@tempa\XKV@documentclass\@nil
  \ifx\XKV@classoptionslist\@empty
    \XTR@writeout{\string\documentclass{\XTR@tempa}}
  \else
    \@temptokena\expandafter{\XKV@classoptionslist}%
    \XTR@writeout{\string\documentclass[\the\@temptokena]{\XTR@tempa}}
  \fi
\fi
```

Perform redefinitions at the beginning of the document.

```
\AtBeginDocument{%
  \ifXTR@handles
    \XTR@writeout{}%
    \XTR@writeout{\string\begin{document}}%
  \fi
```

Redefine environments.

```
  \XKV@ifundefined{XTR@envs}{}{%
    \XKV@for@o\XTR@envs\XTR@tempa{%
```

Check whether the environment is defined.

```
      \XKV@ifundefined\XTR@tempa{%
        \XTR@err{%
          environment ‘\XTR@tempa’ not defined; extraction canceled%
        }%
      }{%
```

Backup the beginning of the environment.

```
        \XTR@namelet{XTR\XTR@tempa}{\XTR@tempa}%
```

Redefine the beginning of the environment. This uses verbatim internally.

```
        \@namedef{\XTR@tempa\expandafter}\expandafter{\expandafter
          \def\expandafter\XTR@tempa\expandafter{\XTR@tempa}%
```

Check whether the current environment should be extracted. Note that \XTR@tempa contains the current environment name.

```
          \XTR@checkxtr{extract}\XTR@tempa
          \ifXTR@extract
```

If extraction is required, write to the target file and to a temporary file for inclusion afterwards.

```
            \XTR@writeout{}\XTR@opentmp
            \@bsphack
            \let\do\@makeother\dospecials\catcode‘\^^M\active
```

\verbatim@processline Process macro for verbatim.

```
            \def\verbatim@processline{%
```

\verbatim@processline is redefined here since the first line is treated specially, see below.

```
              \let\verbatim@processline\XTR@processline@begin
```

Write the content to the files.

```
              \XTR@writeout{%
                \string\begin{\XTR@tempa}\the\verbatim@line
              }%
              \XTR@writetmp{%
                \string\begin{XTR\XTR@tempa}\the\verbatim@line
              }%
            }%

            \XTR@sttrue\let\XTR@tempb\verbatim@
          \else
```

Else, execute the backup of the current environment.

```
            \edef\XTR@tempb{\noexpand\begin{XTR\XTR@tempa}}%
          \fi
          \XTR@tempb
        }%
```

Backup the end of the environment.

```
        \XTR@namelet{endXTR\XTR@tempa}{end\XTR@tempa}%
```

Redefine the end of the environment.

```
        \@namedef{end\XTR@tempa\expandafter}\expandafter{\expandafter
          \def\expandafter\XTR@tempa\expandafter{\XTR@tempa}%
          \ifXTR@extract
            \@esphack
```

Finalize writing and add the \input to the hook at the end of the current environment.

```
            \XTR@writeout{\string\end{\XTR@tempa}}%
            \XTR@writetmp{\string\end{XTR\XTR@tempa}}%
            \XTR@closetmp
            \AfterEndEnv{\input{\jobname.xtr}}%
          \else
```

If not extracting, execute the backup of the end of the environment.

```
            \edef\XTR@tempa{\noexpand\end{XTR\XTR@tempa}}%
            \expandafter\XTR@tempa
          \fi
        }%
      }%
    }%
  }%
```

Redefine commands using the arguments.

```
  \XKV@ifundefined{XTR@cmdsargs}{}{%
```

Once backup the current definitions.

```
    \let\XTR@sect\@sect
    \let\XTR@chapter\@chapter
    \def\XTR@tempb{chapter}%
```

Redefine a list of macros to write themselves to the target file. Chapters and section are treated differently since they are constructed differently. `\chapter*` will not extract itself since this gives technical difficulties due to the fact that this macro is reused at several places inside other macros, taking none-character input in its argument.

```
  \XKV@for@o\XTR@cmdsargs\XTR@tempa{%
    \XKV@ifundefined\XTR@tempa{%
      \XTR@err{command ‘\@backslashchar\XTR@tempa’ not defined;
        extraction canceled%
      }%
    }{%
```

Check whether allowed or not.

```
      \@expandtwoargs\in@{,\XTR@tempa,}%
        {,chapter,section,subsection,subsubsection,}%
      \ifin@
        \ifx\XTR@tempa\XTR@tempb
          \def\@chapter[#1]#2{%
```

Check whether to extract this chapter or not.

```
            \XTR@checkxtr{extract}{chapter}%
            \ifXTR@extract
              \XTR@writeout{}%
              \def\XTR@tempa{#1}%
              \def\XTR@tempb{#2}%
              \ifx\XTR@tempa\XTR@tempb
                \@temptokena{{#2}}%
              \else
                \@temptokena{[#1]{#2}}%
              \fi
```

Write to file.

```
              \XTR@writeout{\string\chapter\the\@temptokena}%
            \fi
```

Typeset the chapter.

```
            \XTR@chapter[#1]{#2}%
          }%
        \else
```

We do a similar thing for sections created with `\@sect`.

```
          \def\@sect#1#2#3#4#5#6[#7]#8{%
            \@expandtwoargs\in@{,#1,}{,\XTR@cmdsargs,}%
            \ifin@
              \XTR@checkxtr{extract}{#1}%
              \ifXTR@extract
                \XTR@writeout{}%
                \def\XTR@tempa{#7}%
                \def\XTR@tempb{#8}%
                \ifx\XTR@tempa\XTR@tempb
                  \@temptokena{{#8}}%
                \else
                  \@temptokena{[#7]{#8}}%
                \fi
                \XTR@writeout{\expandafter
                  \string\csname#1\endcsname\the\@temptokena}%
              \fi
          \fi
          \XTR@sect{#1}{#2}{#3}{#4}{#5}{#6}[#7]{#8}%
        }%
      \fi
    \else
      \XTR@err{unsupported command ‘\XTR@tempa’;
        try the ‘extract-cmdline option}%
    \fi
  }%
}%
}%
\XKV@ifundefined{XTR@cmdsline}{}{%
```

Redefine a list of commands to write themselves and the text on the same line to the target file. This works similar to `\extractline`.

```
  \XKV@for@o\XTR@cmdsline\XTR@tempa{%
    \XKV@ifundefined\XTR@tempa{%
```

Check whether the command is defined.

```
      \XTR@err{command ‘\@backslashchar\XTR@tempa’ not defined;
        extraction canceled}%
    }{%
```

Check whether allowed or not.

```
      \@expandtwoargs\in@{,\XTR@tempa,}%
        {,chapter,section,subsection,subsubsection,}%
      \ifin@
        \XTR@err{%
          use the ‘extract-cmd’ option for command ‘\XTR@tempa’%
        }%
      \else
```

Backup the command.

```
        \XTR@namelet{XTR\XTR@tempa}{\XTR@tempa}%
```

Redefine the command. Note that, inside the definition of the command, `\XTR@tempa` contains the command name.

```
        \@namedef{\XTR@tempa\expandafter}\expandafter{\expandafter
          \def\expandafter\XTR@tempa\expandafter{\XTR@tempa}%
```

Check whether this command should be extracted.

```
          \XTR@checkxtr{extract}\XTR@tempa
            \begingroup
              \let\do\@makeother\dospecials\catcode‘\^^M\active
              \XTR@extractcmdline
        }%
      \fi
    }%
  }%
  \begingroup
    \catcode‘\~=\active\lccode‘\~=‘\^^M
```

`\XTR@extractcmdline` ⟨*text*⟩⟨*eol*⟩

Workhorse for the command line extraction method. This macros reads until the next end of line and saves the content in `\XTR@tempb`.

```
  \lowercase{\endgroup
```

```
      \def\XTR@extractcmdline#1~{\verbatim@line{#1}\XTR@@xtractcmdline}%
    }%
```

`\XTR@@xtractcmdline` Finalize the operation with the content of the current line. We write it to a target file and to a temporary file for execution in the current document. Note that `\XTR@tempa` still contains the current command name.

```
    \def\XTR@@xtractcmdline{%
        \XTR@writeout{}%
        \XTR@writeout{\expandafter\string\csname\XTR@tempa
          \endcsname\the\verbatim@line
        }%
        \XTR@opentmp
        \XTR@writetmp{\expandafter\string\csname XTR\XTR@tempa
          \endcsname\the\verbatim@line
        }%
        \XTR@closetmp
      \endgroup
      \input{\jobname.xtr}%
    }%
  }%
}
```

Finalize writing the target file.

```
\AtEndDocument{%
  \ifXTR@handles
    \XTR@writeout{}%
    \XTR@writeout{\string\end{document}}%
  \fi
  \immediate\closeout\XTR@out
}
\fi
⟨/extract⟩
```

References

[1] Hendri Adriaens. xkeyval package. `CTAN:/macros/latex/contrib/xkeyval`.

[2] Donald Arseneau. selectp package, v0.9. `CTAN:/macros/latex/contrib/misc`, 1992/09/25.

[3] Donald Arseneau. optional package, v2.2. `CTAN:/macros/latex/contrib/misc`, 2001/09.

[4] Stephen Bellantoni. version package. `CTAN:/macros/latex/contrib/misc`, 1990.

[5] Johannes Braams, David Carlisle, Alan Jeffrey, Leslie Lamport, Frank Mittelbach, Chris Rowley, and Rainer Schöpf. The LaTeX 2_ε sources. `CTAN:/macros/latex/base`, 2003.

[6] Victor Eijkhout. comment package, v3.6. `CTAN:/macros/latex/contrib/comment`, 1999/10.

[7] Carsten Heinz. listings package, v1.3. `CTAN:/macros/latex/contrib/listings`, 2004/09/07.

[8] Uwe Lück. versions package, v0.51. `CTAN:/macros/latex/contrib/versions`, 2003/10/15.

[9] Dan Luecking. excludeonly package, v1.0. `CTAN:/macros/latex/contrib/misc`, 2003/03/14.

[10] Andreas Matthias. pdfpages package, v0.3e. `CTAN:/macros/latex/contrib/pdfpages`, 2004/01/31.

[11] Heiko Oberdiek. pagesel package, v1.1. `CTAN:/macros/latex/contrib/oberdiek`, 1999/04/13.

[12] Pablo A. Straub. askinclude package, v1.2e. `CTAN:/macros/latex/contrib/misc`, 1994/11/11.

[13] Rainer Schöpf. verbatim package, v1.5q. `CTAN:/macros/latex/required/tools`, 2003/08/22.

[14] Timothy Van Zandt. xcomment package, v1.2. `CTAN:/macros/latex/contrib/seminar`, 1993/02/12.

[15] Timothy Van Zandt. fancyvrb package, v2.6. `CTAN:/macros/latex/contrib/fancyvrb`, 1998/07/17.

Acknowledgements

Thanks go to David Carlisle for proposing the technique implemented in this package to solve my initial problem.[7] Thanks also go to Michael Kohlhase and Johannes Luber for reporting bugs and providing ideas for extensions of the package.

Version history

v1.0 *(2004/12/19)*

General: Initial release 1

v1.1 *(2005/01/01)*

General: Added conditional extraction using labels 1
  Revised documentation 1
`\verbatim@processline`: Made to write content on same line as environment heading if so in source file 22

v1.2 *(2005/01/06)*

General: Fixed `\AfterEndEnv` hook for nested environments 1
  Increased efficiency of the package 1
  Simplified optional argument check for `extract` environment 1

v1.3 *(2005/01/12)*

General: Added `\extractline` 1
  Added `document-handles` option 1
  Added examples and documentation 1
  Added line extraction method for commands 1

[7] See section 1.

Removed some input checks for more flexibility 1
Simplified package 1
Updated license information 1
`\extractionlabel`: Was `\extractlabel` 15

v1.4 *(2005/02/06)*
General: Added `extractskip` environment 1
Adjusted options section for xkeyval (`\XKV@tkey` not defined anymore) 1
Allow for any filename 1
Changed temp extension to `.xtr` 1
`\XTR@extractcmdline`: Changed temp macro to token register 24

v1.5 *(2005/02/08)*
General: Avoid using counters for conditional extraction with numbers 1
Solved bug in options section 1

v1.6 *(2005/02/14)*
General: Added optional header for extracted files 1
Improved options section 1

v1.7 *(2005/03/31)*
General: Revised documentation 1
Solved small bug with ‘no-header’ option 1

v1.8 *(2005/05/07)*
General: Added `copydocumentclass` option 1
Changed name of option `document-handles` to `handles` 1
Changed name of option `no-header` to `header` 1
Revised documentation and examples 1
`\XTR@processline@write`: Solved bug of not copying empty lines to the target file . . . 18

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.