`latexindent.pl`

`Version 3.24.5`

Chris Hughes [*]

2025-03-13

`latexindent.pl` is a `Perl` script that indents `.tex` (and other) files according to an indentation scheme that the user can modify to suit their taste. Environments, including those with alignment delimiters (such as `tabular`), and commands, including those that can split braces and brackets across lines, are *usually* handled correctly by the script. Options for `verbatim`-like environments and commands, together with indentation after headings (such as `chapter`, `section`, etc) are also available. The script also has the ability to modify line breaks, and to add comment symbols and blank lines; furthermore, it permits string or regex-based substitutions. All user options are customisable via the switches and the YAML interface.

tl;dr, a quick start guide is given in Section 1.3 on page 5.

Contents

---

[*] and contributors! See Section 11.5 on page 155. For all communication, please visit [36].

Section 1

Introduction

1.1 Thanks

I first created `latexindent.pl` to help me format chapter files in a big project. After I blogged about it on the TeX stack exchange [29] I received some positive feedback and follow-up feature requests. A big thank you to Harish Kumar [2] who helped to develop and test the initial versions of the script.

The `YAML`-based interface of `latexindent.pl` was inspired by the wonderful `arara` tool; any similarities are deliberate, and I hope that it is perceived as the compliment that it is. Thank you to Paulo Cereda and the team for releasing this awesome tool; I initially worried that I was going to have to make a GUI for `latexindent.pl`, but the release of `arara` has meant there is no need.

There have been several contributors to the project so far (and hopefully more in the future!); thank you very much to the people detailed in Section 11.5 on page 155 for their valued contributions, and thank you to those who report bugs and request features at [36].

1.2 License

`latexindent.pl` is free and open source, and it always will be; it is released under the GNU General Public License v3.0.

Before you start using it on any important files, bear in mind that `latexindent.pl` has the option to overwrite your `.tex` files. It will always make at least one backup (you can choose how many it makes, see page 28) but you should still be careful when using it. The script has been tested on many files, but there are some known limitations (see Section 10). You, the user, are responsible for ensuring that you maintain backups of your files before running `latexindent.pl` on them. I think it is important at this stage to restate an important part of the license here:

> *This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.*

There is certainly no malicious intent in releasing this script, and I do hope that it works as you expect it to; if it does not, please first of all make sure that you have the correct settings, and then feel free to let me know at [36] with a complete minimum working example as I would like to improve the code as much as possible.

Warning!

Before you try the script on anything important (like your thesis), test it out on the sample files in the `test-case` directory [36].

If you have used any version 2. of `latexindent.pl`, there are a few changes to the interface; see Section M on page 180 and the comments throughout this document for details.*

1.3 Quick start

When `latexindent.pl` reads and writes files, the files are read and written in UTF-8 format by default. That is to say, the encoding format for tex and yaml files needs to be in UTF-8 format.

If you'd like to get started with `latexindent.pl` then simply type

```
cmh:~$ latexindent.pl myfile.tex
```

from the command line.

We give an introduction to `latexindent.pl` using Listing 1; there is no explanation in this section, which is deliberate for a quick start. The rest of the manual is more verbose.

LISTING 1: `quick-start.tex`

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
    demonstration for latexindent.pl.
    \begin{myenv}
      The body of environments and
      other code blocks
        receive indentation.
    \end{myenv}
\end{document}
```

Running

```
cmh:~$ latexindent.pl quick-start.tex
```

gives Listing 2.

LISTING 2: `quick-start-default.tex`

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
   The body of environments and
   other code blocks
   receive indentation.
\end{myenv}
\end{document}
```

example 1 Running

```
cmh:~$ latexindent.pl -l quick-start1.yaml quick-start.tex
```

gives Listing 3. ■

LISTING 3: `quick-start-mod1.tex`

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A␣quick␣start
demonstration␣for␣latexindent.pl.
\begin{myenv}
␣The␣body␣of␣environments␣and
␣other␣code␣blocks
␣receive␣indentation.
\end{myenv}
\end{document}
```

LISTING 4: `quick-start1.yaml`

```
defaultIndent: " "
```

See Section 5.4. ■

example 2 Running

```
cmh:~$ latexindent.pl -l quick-start2.yaml quick-start.tex
```

gives Listing 5.

LISTING 5: `quick-start-mod2.tex`

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A␣quick␣start
demonstration␣for␣latexindent.pl.
\begin{myenv}
␣␣␣The␣body␣of␣environments␣and
␣␣␣other␣code␣blocks
␣␣␣receive␣indentation.
\end{myenv}
\end{document}
```

LISTING 6: `quick-start2.yaml`

```
indentRules:
  myenv: "   "
```

See Section 5.8. ■

example 3 Running

```
cmh:~$ latexindent.pl -l quick-start3.yaml quick-start.tex
```

gives Listing 7. ■

LISTING 7: quick-start-mod3.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
The body of environments and
other code blocks
receive indentation.
\end{myenv}
\end{document}
```

LISTING 8: quick-start3.yaml

```
noAdditionalIndent:
  myenv: 1
```

See Section 5.8. ■

example 4 Running

```
cmh:~$ latexindent.pl -m -l quick-start4.yaml quick-start.tex
```

gives Listing 9.

LISTING 9: quick-start-mod4.tex

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
   The body of environments and
   other code blocks
   receive indentation.
\end{myenv}
\end{document}
```

LISTING 10: quick-start4.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 20
```

Full details of text wrapping in Section 6.1. ■

example 5 Running

```
cmh:~$ latexindent.pl -m -l quick-start5.yaml quick-start.tex
```

gives Listing 11. ■

LISTING 11: `quick-start-mod5.tex`

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
   The body of
   environments and
   other code blocks
   receive
   indentation.
\end{myenv}
\end{document}
```

LISTING 12: `quick-start5.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 20
        blocksFollow:
           other: '\\begin\{myenv\}'
```

Full details of text wrapping in Section 6.1. ■

example 6 Running

```
cmh:~$ latexindent.pl -m -l quick-start6.yaml quick-start.tex
```

gives Listing 13.

LISTING 13: `quick-start-mod6.tex`

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}\begin{document}
A quick start
demonstration for
    latexindent.pl.\begin{myenv}
   The body of environments and
   other code blocks
   receive indentation.
\end{myenv}
\end{document}
```

LISTING 14: `quick-start6.yaml` -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: -1
```

This is an example of a *poly-switch*; full details of *poly-switches* are covered in Section 6.3. ■

example 7 Running

```
cmh:~$ latexindent.pl -m -l quick-start7.yaml quick-start.tex
```

gives Listing 15. ■

LISTING 15: `quick-start-mod7.tex`

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
   The body of environments and
   other code blocks
   receive
    indentation.\end{myenv}\end{document}
```

LISTING 16: `quick-start7.yaml` -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: -1
```

Full details of *poly-switches* are covered in Section 6.3. ■

example 8 Running

```
cmh:~$ latexindent.pl -l quick-start8.yaml quick-start.tex
```

gives Listing 17; note that the *preamble* has been indented.

LISTING 17: `quick-start-mod8.tex`

```
\documentclass{article}
\usepackage[
   inner=2.5cm,
]{geometry}
\begin{document}
A quick start
demonstration for latexindent.pl.
\begin{myenv}
   The body of environments and
   other code blocks
   receive indentation.
\end{myenv}
\end{document}
```

LISTING 18: `quick-start8.yaml`

```
indentPreamble: 1
```

See Section 5.3. ■

example 9 Running

```
cmh:~$ latexindent.pl -l quick-start9.yaml quick-start.tex
```

gives Listing 19. ■

Listing 19: `quick-start-mod9.tex`

```
\documentclass{article}
\usepackage[
inner=2.5cm,
]{geometry}
\begin{document}
   A quick start
   demonstration for latexindent.pl.
   \begin{myenv}
      The body of environments and
      other code blocks
      receive indentation.
   \end{myenv}
\end{document}
```

Listing 20: `quick-start9.yaml`

```
noAdditionalIndent:
  document: 0
```

See Section 5.8. ■

1.4 Required perl modules

If you receive an error message such as that given in Listing 21, then you need to install the missing perl modules.

Listing 21: Possible error messages

```
Can't␣locate␣File/HomeDir.pm␣in␣@INC␣(@INC␣contains:␣
    /Library/Perl/5.12/darwin-thread-multi-2level␣/Library/Perl/5.12␣
    /Network/Library/Perl/5.12/darwin-thread-multi-2level␣
    /Network/Library/Perl/5.12␣
    /Library/Perl/Updates/5.12.4/darwin-thread-multi-2level␣
    /Library/Perl/Updates/5.12.4␣
    /System/Library/Perl/5.12/darwin-thread-multi-2level␣/System/Library/Perl/5.12␣
    /System/Library/Perl/Extras/5.12/darwin-thread-multi-2level␣
    /System/Library/Perl/Extras/5.12␣.)␣at␣helloworld.pl␣line␣10.
BEGIN␣failed--compilation␣aborted␣at␣helloworld.pl␣line␣10.
```

`latexindent.pl` ships with a script to help with this process; if you run the following script, you should be prompted to install the appropriate modules.

```
cmh:~$ perl latexindent-module-installer.pl
```

You might also like to see https://stackoverflow.com/questions/19590042/error-cant-locate-file-homedir-pm-in-inc, for example, as well as Section A on page 157.

1.5 About this documentation

As you read through this documentation, you will see many listings; in this version of the documentation, there are a total of 629. This may seem a lot, but I deem it necessary in presenting the various different options of `latexindent.pl` and the associated output that they are capable of producing.

The different listings are presented using different styles:

Listing 22: `demo-tex.tex`

```
demonstration .tex file
```

This type of listing is a `.tex` file.

Listing 23: `fileExtensionPreference`

```
47 fileExtensionPreference:
48   .tex: 1
49   .sty: 2
50   .cls: 3
51   .bib: 4
```

This type of listing is a `.yaml` file; when you see line numbers given (as here) it means that the snippet is taken directly from `defaultSettings.yaml`, discussed in detail in Section 5 on page 27.

LISTING 24: `modifyLineBreaks`

-m

```
modifyLineBreaks:
  preserveBlankLines: 1                # 0/1
  condenseMultipleBlankLinesInto: 1    # 0/1
```

This type of listing is a `.yaml` file, but it will only be relevant when the `-m` switch is active; see Section 6 on page 79 for more details.

LISTING 25: `replacements`

-r

```
replacements:
  - amalgamate: 1
  - this: latexindent.pl
    that: pl.latexindent
    lookForThis: 0
    when: before
```

This type of listing is a `.yaml` file, but it will only be relevant when the `-r` switch is active; see Section 7 on page 129 for more details.

N: 2017-06-25

You will occasionally see dates shown in the margin (for example, next to this paragraph!) which detail the date of the version in which the feature was implemented; the 'N' stands for 'new as of the date shown' and 'U' stands for 'updated as of the date shown'. If you see ✨, it means that the feature is either new (N) or updated (U) as of the release of the current version; if you see ✨ attached to a listing, then it means that listing is new (N) or updated (U) as of the current version. If you have not read this document before (and even if you have!), then you can ignore every occurrence of the ✨; they are simply there to highlight new and updated features. The new and updated features in this documentation (V3.24.5) are on the following pages:

1.6 A word about regular expressions

As you read this documentation, you may encounter the term *regular expressions*. I've tried to write this documentation in such a way so as to allow you to engage with them or not, as you prefer. This documentation is not designed to be a guide to regular expressions, and if you'd like to read about them, I recommend [35].

SECTION 2

Demonstration: before and after

Let's give a demonstration of some before and after code – after all, you probably won't want to try the script if you don't much like the results. You might also like to watch the video demonstration I made on youtube [49]

As you look at Listings 26 to 31, remember that `latexindent.pl` is just following its rules, and there is nothing particular about these code snippets. All of the rules can be modified so that you can personalise your indentation scheme.

In each of the samples given in Listings 26 to 31 the 'before' case is a 'worst case scenario' with no effort to make indentation. The 'after' result would be the same, regardless of the leading white space at the beginning of each line which is stripped by `latexindent.pl` (unless a `verbatim`-like environment or `noIndentBlock` is specified – more on this in Section 5).

LISTING 26: `filecontents1.tex`

```
\begin{filecontents}{mybib.bib}
@online{strawberryperl,
title="Strawberry Perl",
url="http://strawberryperl.com/"}
@online{cmhblog,
title="A Perl script ...
url="...
}
\end{filecontents}
```

LISTING 27: `filecontents1.tex` default output

```
\begin{filecontents}{mybib.bib}
   @online{strawberryperl,
      title="Strawberry Perl",
      url="http://strawberryperl.com/"}
   @online{cmhblog,
      title="A Perl script ...
      url="...
   }
\end{filecontents}
```

LISTING 28: `tikzset.tex`

```
\tikzset{
shrink inner sep/.code={
\pgfkeysgetvalue...
\pgfkeysgetvalue...
}
}
```

LISTING 29: `tikzset.tex` default output

```
\tikzset{
   shrink inner sep/.code={
         \pgfkeysgetvalue...
         \pgfkeysgetvalue...
      }
}
```

LISTING 30: `pstricks.tex`

```
\def\Picture#1{%
\def\stripH{#1}%
\begin{pspicture}[showgrid]
\psforeach{\row}{%
{{3,2.8,2.7,3,3.1}},%
{2.8,1,1.2,2,3},%
...
}{%
\expandafter...
}
\end{pspicture}}
```

LISTING 31: `pstricks.tex` default output

```
\def\Picture#1{%
   \def\stripH{#1}%
   \begin{pspicture}[showgrid]
      \psforeach{\row}{%
         {{3,2.8,2.7,3,3.1}},%
         {2.8,1,1.2,2,3},%
         ...
      }{%
         \expandafter...
      }
   \end{pspicture}}
```

Section 3

How to use the script

`latexindent.pl` ships as part of the TeXLive distribution for Linux and Mac users; `latexindent.exe` ships as part of the TeXLive for Windows users. These files are also available from github [36] should you wish to use them without a TeX distribution; in this case, you may like to read Section B on page 161 which details how the `path` variable can be updated.

In what follows, we will always refer to `latexindent.pl`, but depending on your operating system and preference, you might substitute `latexindent.exe` or simply `latexindent`.

There are two ways to use `latexindent.pl`: from the command line, and using `arara`; we discuss these in Section 3.2 and Section 3.3 respectively. We will discuss how to change the settings and behaviour of the script in Section 5 on page 27.

3.1 Requirements

3.1.1 Perl users

N: 2018-01-13

Perl users will need a few standard Perl modules – see Section A on page 157 for details; in particular, note that a module installer helper script is shipped with `latexindent.pl`.

3.1.2 Windows users without perl

`latexindent.pl` ships with `latexindent.exe` for Windows users, so that you can use the script with or without a Perl distribution.

`latexindent.exe` is available from [36].

MiKTeX users on Windows may like to see [39] for details of how to use `latexindent.exe` without a Perl installation.

3.1.3 Ubuntu Linux users without perl

`latexindent.pl` ships with `latexindent-linux` for Ubuntu Linux users, so that you can use the script with or without a Perl distribution.

N: 2022-10-30

`latexindent-linux` is available from [36].

3.1.4 macOS users without perl

`latexindent.pl` ships with `latexindent-macos` for macOS users, so that you can use the script with or without a Perl distribution.

N: 2022-10-30

`latexindent-macOS` is available from [36].

3.1.5 conda users

Users of `conda` should see the details given in Section E.

3.1.6 docker users

Users of `docker` should see the details given in Section F.

3.2 From the command line

`latexindent.pl` has a number of different switches/flags/options, which can be combined in any way that you like, either in short or long form as detailed below. `latexindent.pl` produces a `.log` file, `indent.log`, every time it is run; the name of the log file can be customised, but we will refer to the log file as `indent.log` throughout this document. There is a base of information that is written to `indent.log`, but other additional information will be written depending on which of the following options are used.

When using `latexindent.pl` in different ways on different systems, the range of characters supported by its switches/flags/options may vary. We discuss these in Section Section K.

N: 2017-06-25

`-v, -version`

```
cmh:~$ latexindent.pl -v
cmh:~$ latexindent.pl --version
```

This will output only the version number to the terminal.

N: 2022-01-08

`-vv, -vversion`

```
cmh:~$ latexindent.pl -vv
cmh:~$ latexindent.pl --vversion
```

This will output *verbose* version details to the terminal, including the location of `latexindent.pl` and `defaultSettings.yaml`.

`-h, -help`

```
cmh:~$ latexindent.pl -h
cmh:~$ latexindent.pl --help
```

As above this will output a welcome message to the terminal, including the version number and available options.

```
cmh:~$ latexindent.pl myfile.tex
```

This will operate on `myfile.tex`, but will simply output to your terminal; `myfile.tex` will not be changed by `latexindent.pl` in any way using this command.

N: 2022-03-25

You can instruct `latexindent.pl` to operate on multiple (batches) of files, for example

```
cmh:~$ latexindent.pl myfile1.tex myfile2.tex
```

Full details are given in Section C on page 163.

`-w, -overwrite`

```
cmh:~$ latexindent.pl -w myfile.tex
cmh:~$ latexindent.pl --overwrite myfile.tex
cmh:~$ latexindent.pl myfile.tex --overwrite
```

This *will* overwrite `myfile.tex`, but it will make a copy of `myfile.tex` first. You can control the name of the extension (default is `.bak`), and how many different backups are made – more on this in Section 5, and in particular see `backupExtension` and `onlyOneBackUp`.

Note that if `latexindent.pl` can not create the backup, then it will exit without touching your original file; an error message will be given asking you to check the permissions of the backup file.

N: 2022-03-25 **-wd, -overwriteIfDifferent**

```
cmh:~$ latexindent.pl -wd myfile.tex
cmh:~$ latexindent.pl --overwriteIfDifferent myfile.tex
cmh:~$ latexindent.pl myfile.tex --overwriteIfDifferent
```

This *will* overwrite `myfile.tex` but only *if the indented text is different from the original*. If the indented text is *not* different from the original, then `myfile.tex` will *not* be overwritten.

All other details from the `-w` switch are relevant here. If you call `latexindent.pl` with both the `-wd` and the `-w` switch, then the `-w` switch will be deactivated and the `-wd` switch takes priority.

-o=output.tex,-outputfile=output.tex

```
cmh:~$ latexindent.pl -o=output.tex myfile.tex
cmh:~$ latexindent.pl myfile.tex -o=output.tex
cmh:~$ latexindent.pl --outputfile=output.tex myfile.tex
cmh:~$ latexindent.pl --outputfile output.tex myfile.tex
```

This will indent `myfile.tex` and output it to `output.tex`, overwriting it (`output.tex`) if it already exists[1].

Note that if `latexindent.pl` is called with both the `-w` and `-o` switches, then `-w` will be ignored and `-o` will take priority (this seems safer than the other way round). The same is true for the `-wd` switch, and the `-o` switch takes priority over it.

Note that using `-o` as above is equivalent to using

```
cmh:~$ latexindent.pl myfile.tex > output.tex
```

N: 2017-06-25

You can call the `-o` switch with the name of the output file *without* an extension; in this case, `latexindent.pl` will use the extension from the original file. For example, the following two calls to `latexindent.pl` are equivalent:

```
cmh:~$ latexindent.pl myfile.tex -o=output
cmh:~$ latexindent.pl myfile.tex -o=output.tex
```

N: 2017-06-25

You can call the `-o` switch using a + symbol at the beginning; this will concatenate the name of the input file and the text given to the `-o` switch. For example, the following two calls to `latexindent.pl` are equivalent:

```
cmh:~$ latexindent.pl myfile.tex -o=+new
cmh:~$ latexindent.pl myfile.tex -o=myfilenew.tex
```

N: 2017-06-25

You can call the `-o` switch using a ++ symbol at the end of the name of your output file; this tells `latexindent.pl` to search successively for the name of your output file concatenated with $0, 1, \ldots$ while the name of the output file exists. For example,

```
cmh:~$ latexindent.pl myfile.tex -o=output++
```

tells `latexindent.pl` to output to `output0.tex`, but if it exists then output to `output1.tex`, and so on.

Calling `latexindent.pl` with simply

---

[1] Users of version 2.* should note the subtle change in syntax

```
cmh:~$ latexindent.pl myfile.tex -o=++
```

tells it to output to `myfile0.tex`, but if it exists then output to `myfile1.tex` and so on.

The + and ++ feature of the `-o` switch can be combined; for example, calling

```
cmh:~$ latexindent.pl myfile.tex -o=+out++
```

tells `latexindent.pl` to output to `myfileout0.tex`, but if it exists, then try `myfileout1.tex`, and so on.

There is no need to specify a file extension when using the ++ feature, but if you wish to, then you should include it *after* the ++ symbols, for example

```
cmh:~$ latexindent.pl myfile.tex -o=+out++.tex
```

See Section M on page 180 for details of how the interface has changed from Version 2.2 to Version 3.0 for this flag.

`-s, -silent`

```
cmh:~$ latexindent.pl -s myfile.tex
cmh:~$ latexindent.pl myfile.tex -s
```

Silent mode: no output will be given to the terminal.

`-t, -trace`

```
cmh:~$ latexindent.pl -t myfile.tex
cmh:~$ latexindent.pl myfile.tex -t
```

Tracing mode: verbose output will be given to `indent.log`. This is useful if `latexindent.pl` has made a mistake and you're trying to find out where and why. You might also be interested in learning about `latexindent.pl`'s thought process – if so, this switch is for you, although it should be noted that, especially for large files, this does affect performance of the script.

`-tt, -ttrace`

```
cmh:~$ latexindent.pl -tt myfile.tex
cmh:~$ latexindent.pl myfile.tex -tt
```

More detailed tracing mode: this option gives more details to `indent.log` than the standard `trace` option (note that, even more so than with `-t`, especially for large files, performance of the script will be affected).

`-l, -local[=myyaml.yaml,other.yaml,...]`

```
cmh:~$ latexindent.pl -l myfile.tex
cmh:~$ latexindent.pl -l=myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l first.yaml,second.yaml,third.yaml myfile.tex
cmh:~$ latexindent.pl -l=first.yaml,second.yaml,third.yaml myfile.tex
cmh:~$ latexindent.pl myfile.tex -l=first.yaml,second.yaml,third.yaml
```

`latexindent.pl` will always load `defaultSettings.yaml` (rhymes with camel) and if it is called with the `-l` switch and it finds `localSettings.yaml` in the same directory as `myfile.tex`, then, if not found, it looks for `localSettings.yaml` (and friends, see Section 4.2 on page 24) in the current

U: 2021-03-14 working directory, then these settings will be added to the indentation scheme. Information will be given in `indent.log` on the success or failure of loading `localSettings.yaml`.

The `-l` flag can take an *optional* parameter which details the name (or names separated by commas) of a YAML file(s) that resides in the same directory as `myfile.tex`; you can use this option if you would like to load a settings file in the current working directory that is *not* called U: 2017-08-21 `localSettings.yaml`. In fact, you can specify both *relative* and *absolute paths* for your YAML files; for example

```
cmh:~$ latexindent.pl -l=../../myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l=/home/cmhughes/Desktop/myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l=C:\Users\cmhughes\Desktop\myyaml.yaml myfile.tex
```

You will find a lot of other explicit demonstrations of how to use the `-l` switch throughout this documentation,

N: 2017-06-25 You can call the `-l` switch with a '+' symbol either before or after another YAML file; for example:

```
cmh:~$ latexindent.pl -l=+myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l "+␣myyaml.yaml" myfile.tex
cmh:~$ latexindent.pl -l=myyaml.yaml+  myfile.tex
```

which translate, respectively, to

```
cmh:~$ latexindent.pl -l=localSettings.yaml,myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l=localSettings.yaml,myyaml.yaml myfile.tex
cmh:~$ latexindent.pl -l=myyaml.yaml,localSettings.yaml myfile.tex
```

Note that the following is *not* allowed:

```
cmh:~$ latexindent.pl -l+myyaml.yaml myfile.tex
```

and

```
cmh:~$ latexindent.pl -l + myyaml.yaml myfile.tex
```

will *only* load `localSettings.yaml`, and `myyaml.yaml` will be ignored. If you wish to use spaces between any of the YAML settings, then you must wrap the entire list of YAML files in quotes, as demonstrated above.

N: 2017-06-25 You may also choose to omit the `yaml` extension, such as

```
cmh:~$ latexindent.pl -l=localSettings,myyaml myfile.tex
```

`-y, -yaml=yaml settings`

```
cmh:~$ latexindent.pl myfile.tex -y="defaultIndent:␣’␣’"
cmh:~$ latexindent.pl myfile.tex -y="defaultIndent:␣’␣’,maximumIndentation:’␣’"
cmh:~$ latexindent.pl myfile.tex -y="indentRules:␣one:␣’\t\t\t\t’"
cmh:~$ latexindent.pl myfile.tex
    -y=’modifyLineBreaks:environments:EndStartsOnOwnLine:3’ -m
cmh:~$ latexindent.pl myfile.tex
    -y=’modifyLineBreaks:environments:one:EndStartsOnOwnLine:3’ -m
```

N: 2017-08-21 You can specify YAML settings from the command line using the `-y` or `-yaml` switch; sample demonstrations are given above. Note, in particular, that multiple settings can be specified by separating them via commas. There is a further option to use a `;` to separate fields, which is demonstrated in Section 4.3 on page 25.

Any settings specified via this switch will be loaded *after* any specified using the `-l` switch. This is discussed further in Section 4.4 on page 25.

`-d, -onlydefault`

```
cmh:~$ latexindent.pl -d myfile.tex
```

Only `defaultSettings.yaml`: you might like to read Section 5 before using this switch. By default, `latexindent.pl` will always search for `indentconfig.yaml` or `.indentconfig.yaml` in your home directory. If you would prefer it not to do so then (instead of deleting or renaming `indentconfig.yaml` or `.indentconfig.yaml`) you can simply call the script with the `-d` switch; note that this will also tell the script to ignore `localSettings.yaml` even if it has been called with the `-l` switch; `latexindent.pl` will also ignore any settings specified from the `-y` switch. U: 2017-08-21

`-c, -cruft=<directory>`

```
cmh:~$ latexindent.pl -c=/path/to/directory/ myfile.tex
```

If you wish to have backup files and `indent.log` written to a directory other than the current working directory, then you can send these ‘cruft’ files to another directory. Note the use of a trailing forward slash.

If the cruft directory does not exist, `latexindent.pl` will attempt to create it.

`-g, -logfile=<name of log file>`

```
cmh:~$ latexindent.pl -g=other.log myfile.tex
cmh:~$ latexindent.pl -g other.log myfile.tex
cmh:~$ latexindent.pl --logfile other.log myfile.tex
cmh:~$ latexindent.pl myfile.tex -g other.log
```

By default, `latexindent.pl` reports information to `indent.log`, but if you wish to change the name of this file, simply call the script with your chosen name after the `-g` switch as demonstrated above.

N: 2021-05-07 If `latexindent.pl` can not open the log file that you specify, then the script will operate, and no log file will be produced; this might be helpful to users who wish to specify the following, for example

```
cmh:~$ latexindent.pl -g /dev/null myfile.tex
```

`-sl, -screenlog`

```
cmh:~$ latexindent.pl -sl myfile.tex
cmh:~$ latexindent.pl -screenlog myfile.tex
```

N: 2018-01-13 Using this option tells `latexindent.pl` to output the log file to the screen, as well as to your chosen log file.

`-m, -modifylinebreaks`

```
cmh:~$ latexindent.pl -m myfile.tex
cmh:~$ latexindent.pl -modifylinebreaks myfile.tex
```

One of the most exciting developments in Version 3.0 is the ability to modify line breaks; for full details see Section 6 on page 79

`latexindent.pl` can also be called on a file without the file extension, for example

```
cmh:~$ latexindent.pl myfile
```

and in which case, you can specify the order in which extensions are searched for; see Listing 35 on page 27 for full details.

`STDIN`

```
cmh:~$ cat myfile.tex | latexindent.pl
cmh:~$ cat myfile.tex | latexindent.pl -
```

N: 2018-01-13

`latexindent.pl` will allow input from STDIN, which means that you can pipe output from other commands directly into the script. For example assuming that you have content in `myfile.tex`, then the above command will output the results of operating upon `myfile.tex`.

If you wish to use this feature with your own local settings, via the `-l` switch, then you should finish your call to `latexindent.pl` with a `-` sign:

```
cmh:~$ cat myfile.tex | latexindent.pl -l=mysettings.yaml -
```

U: 2018-01-13

Similarly, if you simply type `latexindent.pl` at the command line, then it will expect (STDIN) input from the command line.

```
cmh:~$ latexindent.pl
```

Once you have finished typing your input, you can press

- `CTRL+D` on Linux
- `CTRL+Z` followed by `ENTER` on Windows

to signify that your input has finished. Thanks to [9] for an update to this feature.

`-r, -replacement`

```
cmh:~$ latexindent.pl -r myfile.tex
cmh:~$ latexindent.pl -replacement myfile.tex
```

N: 2019-07-13

You can call `latexindent.pl` with the `-r` switch to instruct it to perform replacements/substitutions on your file; full details and examples are given in Section 7 on page 129.

`-rv, -replacementrespectverb`

```
cmh:~$ latexindent.pl -rv myfile.tex
cmh:~$ latexindent.pl -replacementrespectverb myfile.tex
```

N: 2019-07-13

You can instruct `latexindent.pl` to perform replacements/substitutions by using the `-rv` switch, but will *respect verbatim code blocks*; full details and examples are given in Section 7 on page 129.

`-rr, -onlyreplacement`

```
cmh:~$ latexindent.pl -rr myfile.tex
cmh:~$ latexindent.pl -onlyreplacement myfile.tex
```

N: 2019-07-13

You can instruct `latexindent.pl` to skip all of its other indentation operations and *only* perform replacements/substitutions by using the `-rr` switch; full details and examples are given in Section 7 on page 129.

`-k, -check`

```
cmh:~$ latexindent.pl -k myfile.tex
cmh:~$ latexindent.pl -check myfile.tex
```

N: 2021-09-16

You can instruct `latexindent.pl` to check if the text after indentation matches that given in the original file.

The `exit` code of `latexindent.pl` is 0 by default. If you use the `-k` switch then

- if the text after indentation matches that given in the original file, then the exit code is 0;
- if the text after indentation does *not* match that given in the original file, then the exit code is 1.

The value of the exit code may be important to those wishing to, for example, check the status of the indentation in continuous integration tools such as GitHub Actions. Full details of the exit codes of `latexindent.pl` are given in Table 1.

A simple `diff` will be given in `indent.log`.

-kv, -checkv

```
cmh:~$ latexindent.pl -kv myfile.tex
cmh:~$ latexindent.pl -checkv myfile.tex
```

N: 2021-09-16

The `check verbose` switch is exactly the same as the `-k` switch, except that it is *verbose*, and it will output the (simple) diff to the terminal, as well as to `indent.log`.

-n, -lines=MIN-MAX

```
cmh:~$ latexindent.pl -n 5-8 myfile.tex
cmh:~$ latexindent.pl -lines 5-8 myfile.tex
```

N: 2021-09-16

The `lines` switch instructs `latexindent.pl` to operate only on specific line ranges within `myfile.tex`.

Complete demonstrations are given in Section 8.

-GCString

```
cmh:~$ latexindent.pl --GCString myfile.tex
```

N: 2022-03-25

instructs `latexindent.pl` to load the `Unicode::GCString` module. This should only be necessary if you find that the alignment at ampersand routine (described in Section 5.5) does not work for your language. Further details are given in Section A.3.

3.3 From arara

Using `latexindent.pl` from the command line is fine for some folks, but others may find it easier to use from `arara`; you can find the arara rule for `latexindent.pl` and its associated documentation at [1].

3.4 Summary of exit codes

Assuming that you call `latexindent.pl` on `myfile.tex`

```
cmh:~$ latexindent.pl myfile.tex
```

then `latexindent.pl` can exit with the exit codes given in Table 1.

TABLE 1: Exit codes for `latexindent.pl`

exit code	indentation	status
0	✔	success; if `-k` or `-kv` active, indented text matches original
0	✖	success; if `-version`, `-vversion` or `-help`, no indentation performed
1	✔	success, and `-k` or `-kv` active; indented text *different* from original
2	✖	failure, `defaultSettings.yaml` could not be read
3	✖	failure, myfile.tex not found
4	✖	failure, myfile.tex exists but cannot be read
5	✖	failure, `-w` active, and back-up file cannot be written
6	✖	failure, `-c` active, and cruft directory could not be created

SECTION 4

indentconfig.yaml, local settings and the -y switch

The behaviour of `latexindent.pl` is controlled from the settings specified in any of the YAML files that you tell it to load. By default, `latexindent.pl` will only load `defaultSettings.yaml`, but there are a few ways that you can tell it to load your own settings files.

N: 2023-01-01

We focus our discussion on `indentconfig.yaml`, but there are other options which are detailed in Section H.

4.1 indentconfig.yaml and .indentconfig.yaml

`latexindent.pl` will always check your home directory for `indentconfig.yaml` and `.indentconfig.yaml` (unless it is called with the `-d` switch), which is a plain text file you can create that contains the *absolute* paths for any settings files that you wish `latexindent.pl` to load. There is no difference between `indentconfig.yaml` and `.indentconfig.yaml`, other than the fact that `.indentconfig.yaml` is a 'hidden' file; thank you to [5] for providing this feature. In what follows, we will use `indentconfig.yaml`, but it is understood that this could equally represent `.indentconfig.yaml`. If you have both files in existence then `indentconfig.yaml` takes priority.

For Mac and Linux users, their home directory is `/username` while Windows (Vista onwards) is `C:\Users\username`[2] Listing 32 shows a sample `indentconfig.yaml` file.

LISTING 32: `indentconfig.yaml` (sample)

```
# Paths to user settings for latexindent.pl
#
# Note that the settings will be read in the order you
# specify here- each successive settings file will overwrite
# the variables that you specify

paths:
- /home/cmhughes/Documents/yamlfiles/mysettings.yaml
- /home/cmhughes/folder/othersettings.yaml
- /some/other/folder/anynameyouwant.yaml
- C:\Users\chughes\Documents\mysettings.yaml
- C:\Users\chughes\Desktop\test spaces\more spaces.yaml
```

Note that the `.yaml` files you specify in `indentconfig.yaml` will be loaded in the order in which you write them. Each file doesn't have to have every switch from `defaultSettings.yaml`; in fact, I recommend that you only keep the switches that you want to *change* in these settings files.

To get started with your own settings file, you might like to save a copy of `defaultSettings.yaml` in another directory and call it, for example, `mysettings.yaml`. Once you have added the path to `indentconfig.yaml` you can change the switches and add more code-block names to it as you see fit – have a look at Listing 33 for an example that uses four tabs for the default indent, adds the `tabbing` environment/command to the list of environments that contains alignment delimiters; you might also like to refer to the many YAML files detailed throughout the rest of this documentation.

---

[2] If you're not sure where to put `indentconfig.yaml`, don't worry `latexindent.pl` will tell you in the log file exactly where to put it assuming it doesn't exist already.

LISTING 33: `mysettings.yaml` (example)

```
# Default value of indentation
defaultIndent: "\t\t\t\t"

# environments that have tab delimiters, add more
# as needed
lookForAlignDelims:
    tabbing: 1
```

You can make sure that your settings are loaded by checking `indent.log` for details – if you have specified a path that `latexindent.pl` doesn't recognise then you'll get a warning, otherwise you'll get confirmation that `latexindent.pl` has read your settings file [3].

Warning!

When editing `.yaml` files it is *extremely* important to remember how sensitive they are to spaces. I highly recommend copying and pasting from `defaultSettings.yaml` when you create your first `whatevernameyoulike.yaml` file.

If `latexindent.pl` can not read your `.yaml` file it will tell you so in `indent.log`.

N: 2024-04-28

As of you can specify the `paths` field from Listing 32 within any of your `latexindent.yaml` and friends settings files. This can lead to creative nesting of configuration files; a demonstration is given in Section I on page 174.

4.2 localSettings.yaml and friends

U: 2021-03-14

The `-l` switch tells `latexindent.pl` to look for `localSettings.yaml` and/or friends in the *same directory* as `myfile.tex`. For example, if you use the following command

```
cmh:~$ latexindent.pl -l myfile.tex
```

then `latexindent.pl` will search for and then, assuming they exist, load each of the following files in the following order:

1. localSettings.yaml
2. latexindent.yaml
3. .localSettings.yaml
4. .latexindent.yaml

These files will be assumed to be in the same directory as `myfile.tex`, or otherwise in the current working directory. You do not need to have all of the above files, usually just one will be sufficient. In what follows, whenever we refer to `localSettings.yaml` it is assumed that it can mean any of the four named options listed above.

If you'd prefer to name your `localSettings.yaml` file something different, (say, `mysettings.yaml` as in Listing 33) then you can call `latexindent.pl` using, for example,

```
cmh:~$ latexindent.pl -l=mysettings.yaml myfile.tex
```

Any settings file(s) specified using the `-l` switch will be read *after* `defaultSettings.yaml` and, assuming they exist, any user setting files specified in `indentconfig.yaml`.

Your settings file can contain any switches that you'd like to change; a sample is shown in Listing 34, and you'll find plenty of further examples throughout this manual.

---

[3] Windows users may find that they have to end `.yaml` files with a blank line

LISTING 34: `localSettings.yaml` (example)

```
#  verbatim environments - environments specified
#  here will not be changed at all!
verbatimEnvironments:
    cmhenvironment: 0
    myenv: 1
```

You can make sure that your settings file has been loaded by checking `indent.log` for details; if it can not be read then you receive a warning, otherwise you'll get confirmation that `latexindent.pl` has read your settings file.

4.3 The -y|yaml switch

N: 2017-08-21

You may use the `-y` switch to load your settings; for example, if you wished to specify the settings from Listing 34 using the `-y` switch, then you could use the following command:

```
cmh:~$ latexindent.pl -y="verbatimEnvironments:cmhenvironment:0;myenv:1" myfile.tex
```

Note the use of `;` to specify another field within `verbatimEnvironments`. This is shorthand, and equivalent, to using the following command:

```
cmh:~$ latexindent.pl
    -y="verbatimEnvironments:cmhenvironment:0,verbatimEnvironments:myenv:1"
    myfile.tex
```

You may, of course, specify settings using the `-y` switch as well as, for example, settings loaded using the `-l` switch; for example,

```
cmh:~$ latexindent.pl -l=mysettings.yaml
    -y="verbatimEnvironments:cmhenvironment:0;myenv:1" myfile.tex
```

Any settings specified using the `-y` switch will be loaded *after* any specified using `indentconfig.yaml` and the `-l` switch.

If you wish to specify any regex-based settings using the `-y` switch, it is important not to use quotes surrounding the regex; for example, with reference to the 'one sentence per line' feature (Section 6.2 on page 95) and the listings within Listing 375 on page 98, the following settings give the option to have sentences end with a semicolon

```
cmh:~$ latexindent.pl -m
    --yaml=’modifyLineBreaks:oneSentencePerLine:sentencesEndWith:other:\;’
```

Note that the `paths` settings (see Section I on page 174) can *not* be specified using the `-y` switch.

4.4 Settings load order

`latexindent.pl` loads the settings files in the following order:

1. `defaultSettings.yaml` is always loaded, and can not be renamed;
2. `anyUserSettings.yaml` and any other arbitrarily-named files specified in `indentconfig.yaml`;
3. `localSettings.yaml` but only if found in the same directory as `myfile.tex` and called with `-l` switch; this file can be renamed, provided that the call to `latexindent.pl` is adjusted accordingly (see Section 4.2). You may specify both relative and absolute paths to other YAML files using the `-l` switch, separating multiple files using commas; U: 2017-08-21
4. any settings specified in the `-y` switch. N: 2017-08-21

A visual representation of this is given in Figure 1.

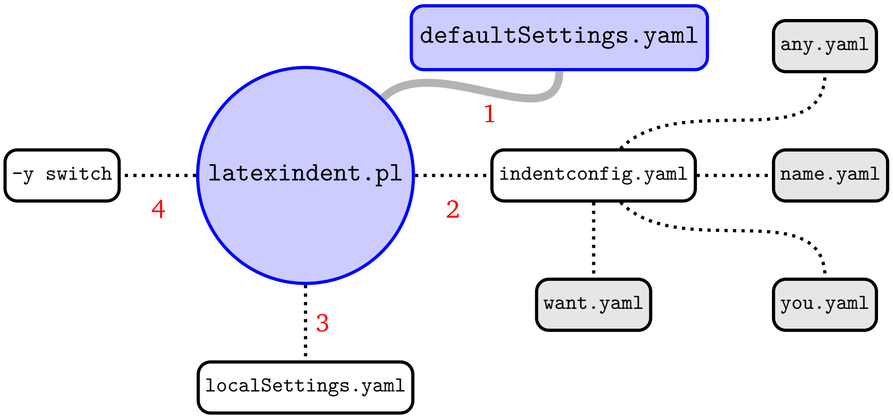

FIGURE 1: Schematic of the load order described in Section 4.4; solid lines represent mandatory files, dotted lines represent optional files. `indentconfig.yaml` can contain as many files as you like. The files will be loaded in order; if you specify settings for the same field in more than one file, the most recent takes priority.

SECTION 5

defaultSettings.yaml

`latexindent.pl` loads its settings from `defaultSettings.yaml`. The idea is to separate the behaviour of the script from the internal working – this is very similar to the way that we separate content from form when writing our documents in LaTeX.

If you look in `defaultSettings.yaml` you'll find the switches that govern the behaviour of `latexindent.pl`. If you're not sure where `defaultSettings.yaml` resides on your computer, don't worry as `indent.log` will tell you where to find it. `defaultSettings.yaml` is commented, but here is a description of what each switch is designed to do. The default value is given in each case; whenever you see *integer* in *this* section, assume that it must be greater than or equal to `0` unless otherwise stated.

For most of the settings in `defaultSettings.yaml` that are specified as integers, then we understand `0` to represent 'off' and `1` to represent 'on'. For fields that allow values other than `0` or `1`, it is hoped that the specific context and associated commentary should make it clear which values are allowed.

`fileExtensionPreference`: ⟨*fields*⟩

`latexindent.pl` can be called to act on a file without specifying the file extension. For example we can call

```
cmh:~$ latexindent.pl myfile
```

in which case the script will look for `myfile` with the extensions specified in `fileExtensionPreference` in their numeric order. If no match is found, the script will exit. As with all of the fields, you should change and/or add to this as necessary.

LISTING 35: `fileExtensionPreference`

```
fileExtensionPreference:
  .tex: 1
  .sty: 2
  .cls: 3
  .bib: 4
```

Calling `latexindent.pl myfile` with the (default) settings specified in Listing 35 means that the script will first look for `myfile.tex`, then `myfile.sty`, `myfile.cls`, and finally `myfile.bib` in order[4].

5.1 Backup and log file preferences

`backupExtension`: ⟨*extension name*⟩

If you call `latexindent.pl` with the `-w` switch (to overwrite `myfile.tex`) then it will create a backup file before doing any indentation; the default extension is `.bak`, so, for example, `myfile.bak0` would be created when calling `latexindent.pl myfile.tex` for the first time.

By default, every time you subsequently call `latexindent.pl` with the `-w` to act upon `myfile.tex`, it will create successive back up files: `myfile.bak1`, `myfile.bak2`, etc.

---

[4] Throughout this manual, listings shown with line numbers represent code taken directly from `defaultSettings.yaml`.

`onlyOneBackUp`: ⟨*integer*⟩

If you don't want a backup for every time that you call `latexindent.pl` (so you don't want `myfile.bak1`, `myfile.bak2`, etc) and you simply want `myfile.bak` (or whatever you chose `backupExtension` to be) then change `onlyOneBackUp` to `1`; the default value of `onlyOneBackUp` is `0`.

`maxNumberOfBackUps`: ⟨*integer*⟩

Some users may only want a finite number of backup files, say at most 3, in which case, they can change this switch. The smallest value of `maxNumberOfBackUps` is 0 which will *not* prevent backup files being made; in this case, the behaviour will be dictated entirely by `onlyOneBackUp`. The default value of `maxNumberOfBackUps` is `0`.

`cycleThroughBackUps`: ⟨*integer*⟩

Some users may wish to cycle through backup files, by deleting the oldest backup file and keeping only the most recent; for example, with `maxNumberOfBackUps: 4`, and `cycleThroughBackUps` set to `1` then the `copy` procedure given below would be obeyed.

```
cmh:~$ copy myfile.bak1 to myfile.bak0
cmh:~$ copy myfile.bak2 to myfile.bak1
cmh:~$ copy myfile.bak3 to myfile.bak2
cmh:~$ copy myfile.bak4 to myfile.bak3
```

The default value of `cycleThroughBackUps` is `0`.

`logFilePreferences`: ⟨*fields*⟩

`latexindent.pl` writes information to `indent.log`, some of which can be customized by changing `logFilePreferences`; see Listing 36. If you load your own user settings (see Section 4 on page 23) then `latexindent.pl` will detail them in `indent.log`; you can choose not to have the details logged by switching `showEveryYamlRead` to `0`. Once all of your settings have been loaded, you can see the amalgamated settings in the log file by switching `showAmalgamatedSettings` to `1`, if you wish.

LISTING 36: `logFilePreferences`

```
logFilePreferences:
  showEveryYamlRead: 1
  showAmalgamatedSettings: 0
  showDecorationStartCodeBlockTrace: 0
  showDecorationFinishCodeBlockTrace: 0
  endLogFileWith: '--------------'
  showGitHubInfoFooter: 1
  Dumper:
    Terse: 1
    Indent: 1
    Useqq: 1
    Deparse: 1
    Quotekeys: 0
    Sortkeys: 1
    Pair: " => "
```

N: 2018-01-13 When either of the `trace` modes (see page 17) are active, you will receive detailed information in `indent.log`. You can specify character strings to appear before and after the notification of a found code block using, respectively, `showDecorationStartCodeBlockTrace` and `showDecorationFinishCodeBlockTra` A demonstration is given in Section J on page 177.

The log file will end with the characters given in `endLogFileWith`, and will report the `GitHub` address of `latexindent.pl` to the log file if `showGitHubInfoFooter` is set to `1`.

U: 2021-03-14 Note: `latexindent.pl` no longer uses the `log4perl` module to handle the creation of the logfile.

U: 2021-06-19 Some of the options for Perl's `Dumper` module can be specified in Listing 36; see [34] and [33] for more information. These options will mostly be helpful for those calling `latexindent.pl` with the `-tt` option described in Section 3.2.

5.2 Verbatim code blocks

`verbatimEnvironments`: ⟨*fields*⟩

A field that contains a list of environments that you would like left completely alone – no indentation will be performed on environments that you have specified in this field, see Listing 37.

Listing 37: `verbatimEnvironments`

```
verbatimEnvironments:
  verbatim: 1
  lstlisting: 1
  minted: 1
```

Listing 38: `verbatimCommands`

```
verbatimCommands:
  verb: 1
  lstinline: 1
```

Note that if you put an environment in `verbatimEnvironments` and in other fields such as `lookForAlignDelims` or `noAdditionalIndent` then `latexindent.pl` will *always* prioritize `verbatimEnvironments`.

N: 2021-10-30 You can, optionally, specify the `verbatim` field using the `name` field which takes a regular expression as its argument; thank you to [18] for contributing this feature.

example 10 For demonstration, then assuming that your file contains the environments `latexcode`, `latexcode*`, `pythoncode` and `pythoncode*`, then the listings given in Listings 39 and 40 are equivalent.

Listing 39: `nameAsRegex1.yaml`

```
verbatimEnvironments:
    latexcode: 1
    latexcode*: 1
    pythoncode: 1
    pythoncode*: 1
```

Listing 40: `nameAsRegex2.yaml`

```
verbatimEnvironments:
    nameAsRegex:
      name: '\w+code\*?'
      lookForThis: 1
```

With reference to Listing 40:

- the `name` field as specified here means *any word followed by the word code, optionally followed by* *;
- we have used `nameAsRegex` to identify this field, but you can use any description you like;
- the `lookForThis` field is optional, and can take the values 0 (off) or 1 (on); by default, it is assumed to be 1 (on).

■

`verbatimCommands`: ⟨*fields*⟩

A field that contains a list of commands that are verbatim commands, for example `\lstinline`; any commands populated in this field are protected from line breaking routines (only relevant if the `-m` is active, see Section 6 on page 79).

With reference to Listing 38, by default `latexindent.pl` looks for `\verb` immediately followed by another character, and then it takes the body as anything up to the next occurrence of the character; this means that, for example, `\verb!x+3!` is treated as a `verbatimCommands`.

N: 2021-10-30 You can, optionally, specify the `verbatimCommands` field using the `name` field which takes a regular expression as its argument; thank you to [18] for contributing this feature.

example 11 For demonstration, then assuming that your file contains the commands `verbinline`, `myinline` then the listings given in Listings 41 and 42 are equivalent.

LISTING 41: `nameAsRegex3.yaml`

```
verbatimCommands:
    verbinline: 1
    myinline: 1
```

LISTING 42: `nameAsRegex4.yaml`

```
verbatimCommands:
    nameAsRegex:
      name: ’\w+inline’
      lookForThis: 1
```

With reference to Listing 42:

- the `name` field as specified here means *any word followed by the word inline*;
- we have used `nameAsRegex` to identify this field, but you can use any description you like;
- the `lookForThis` field is optional, and can take the values 0 (off) or 1 (on); by default, it is assumed to be 1 (on). ■

`noIndentBlock`: ⟨*fields*⟩

If you have a block of code that you don’t want `latexindent.pl` to touch (even if it is *not* a verbatim-like environment) then you can wrap it in an environment from `noIndentBlock`; you can use any name you like for this, provided you populate it as demonstrate in Listing 43.

LISTING 43: `noIndentBlock`

```
noIndentBlock:
  noindent: 1
  cmhtest: 1
```

Of course, you don’t want to have to specify these as null environments in your code, so you use them with a comment symbol, `%`, followed by as many spaces (possibly none) as you like; see Listing 44 for example.

LISTING 44: `noIndentBlock.tex`

```
% \begin{noindent}
some before text
        this code
                won’t
     be touched
                    by
             latexindent.pl!
some after text
% \end{noindent}
```

Important note: it is assumed that the `noindent` block statements specified in this way appear on their own line.

example 12 The `noIndentBlock` fields can also be specified in terms of `begin` and `end` fields. We use the code in Listing 45 to demonstrate this feature.

N: 2021-06-19

LISTING 45: `noIndentBlock1.tex`

```
some before text
        this code
                won’t
     be touched
                    by
             latexindent.pl!
some after text
```

The settings given in Listings 46 and 47 are equivalent: ■

Listing 46: `noindent1.yaml`

```
noIndentBlock:
    demo:
        begin: 'some\hbefore'
        body: '.*?'
        end: 'some\hafter\htext'
        lookForThis: 1
```

Listing 47: `noindent2.yaml`

```
noIndentBlock:
    demo:
        begin: 'some\hbefore'
        end: 'some\hafter\htext'
```

Listing 48: `noindent3.yaml`

```
noIndentBlock:
    demo:
        begin: 'some\hbefore'
        body: '.*?'
        end: 'some\hafter\htext'
        lookForThis: 0
```

Upon running the commands

```
cmh:~$ latexindent.pl -l noindent1.yaml noindent1
cmh:~$ latexindent.pl -l noindent2.yaml noindent1
```

then we receive the output given in Listing 49.

Listing 49: `noIndentBlock1.tex` using Listing 46 or Listing 47

```
some before text
        this code
                won't
     be touched
                    by
             latexindent.pl!
some after text
```

■

The `begin`, `body` and `end` fields for `noIndentBlock` are all *regular expressions*. If the `body` field is not specified, then it takes a default value of `.*?` which is written explicitly in Listing 46. In this context, we interpret `.*?` in words as *the fewest number of characters (possibly none) until the 'end' field is reached*.

The `lookForThis` field is optional, and can take the values 0 (off) or 1 (on); by default, it is assumed to be 1 (on).

example 13 Using Listing 48 demonstrates setting `lookForThis` to 0 (off); running the command

```
cmh:~$ latexindent.pl -l noindent3.yaml noindent1
```

gives the output in Listing 50.

Listing 50: `noIndentBlock1.tex` using Listing 48

```
some before text
this code
won't
be touched
by
latexindent.pl!
some after text
```

■

We will demonstrate this feature later in the documentation in Listing 582.

N: 2021-10-30

You can, optionally, specify the `noIndentBlock` field using the `name` field which takes a regular expression as its argument; thank you to [18] for contributing this feature.

example 14 For demonstration, then assuming that your file contains the environments `testnoindent`, `testnoindent*` then the listings given in Listings 51 and 52 are equivalent.

■

LISTING 51: `nameAsRegex5.yaml`

```yaml
noIndentBlock:
    mytest:
        begin: '\\begin\{testnoindent\*?\}'
        end: '\\end\{testnoindent\*?\}'
```

LISTING 52: `nameAsRegex6.yaml`

```yaml
noIndentBlock:
    nameAsRegex:
        name: '\w+noindent\*?'
        lookForThis: 1
```

With reference to Listing 52:

- the `name` field as specified here means *any word followed by the word noindent, optionally followed by* *;
- we have used `nameAsRegex` to identify this field, but you can use any description you like;
- the `lookForThis` field is optional, and can take the values 0 (off) or 1 (on); by default, it is assumed to be 1 (on).

■

5.3 filecontents and preamble

`fileContentsEnvironments`: ⟨*field*⟩

Before `latexindent.pl` determines the difference between preamble (if any) and the main document, it first searches for any of the environments specified in `fileContentsEnvironments`, see Listing 53. The behaviour of `latexindent.pl` on these environments is determined by their location (preamble or not), and the value `indentPreamble`, discussed next.

LISTING 53: `fileContentsEnvironments`

```yaml
fileContentsEnvironments:
  filecontents: 1
  filecontents*: 1
```

`indentPreamble`: **0|1**

The preamble of a document can sometimes contain some trickier code for `latexindent.pl` to operate upon. By default, `latexindent.pl` won't try to operate on the preamble (as `indentPreamble` is set to `0`, by default), but if you'd like `latexindent.pl` to try then change `indentPreamble` to `1`.

`lookForPreamble`: ⟨*fields*⟩

Not all files contain preamble; for example, `sty`, `cls` and `bib` files typically do *not*. Referencing Listing 54, if you set, for example, `.tex` to `0`, then regardless of the setting of the value of `indentPreamble`, preamble will not be assumed when operating upon `.tex` files.

LISTING 54: lookForPreamble

```yaml
lookForPreamble:
  .tex: 1
  .sty: 0
  .cls: 0
  .bib: 0
  STDIN: 1
```

`preambleCommandsBeforeEnvironments`: **0|1**

Assuming that `latexindent.pl` is asked to operate upon the preamble of a document, when this switch is set to `0` then environment code blocks will be sought first, and then command code blocks. When this switch is set to `1`, commands will be sought first. The example that first motivated this switch contained the code given in Listing 55.

LISTING 55: Motivating `preambleCommandsBeforeEnvironments`

```
...
preheadhook={\begin{mdframed}[style=myframedstyle]},
postfoothook=\end{mdframed},
...
```

5.4 Indentation and horizontal space

`defaultIndent`: ⟨*horizontal space*⟩

This is the default indentation used in the absence of other details for the code block with which we are working. The default value is `\t` which means a tab; we will explore customisation beyond `defaultIndent` in Section 5.8 on page 55.

If you're interested in experimenting with `latexindent.pl` then you can *remove* all indentation by setting `defaultIndent: ""`.

`removeTrailingWhitespace`: ⟨*fields*⟩

Trailing white space can be removed both *before* and *after* processing the document, as detailed in Listing 56; each of the fields can take the values `0` or `1`. See Listings 470 to 472 on page 116 for before and after results. Thanks to [3] for providing this feature.

LISTING 56: removeTrailingWhitespace

```
removeTrailingWhitespace:
  beforeProcessing: 0
  afterProcessing: 1
```

LISTING 57: removeTrailingWhitespace (alt)

```
removeTrailingWhitespace: 1
```

N: 2017-06-28

You can specify `removeTrailingWhitespace` simply as `0` or `1`, if you wish; in this case, `latexindent.pl` will set both `beforeProcessing` and `afterProcessing` to the value you specify; see Listing 57.

5.5 Aligning at delimiters

`lookForAlignDelims`: ⟨*fields*⟩

This contains a list of code blocks that are operated upon in a special way by `latexindent.pl` (see Listing 58). In fact, the fields in `lookForAlignDelims` can actually take two different forms: the *basic* version is shown in Listing 58 and the *advanced* version in Listing 61; we will discuss each in turn.

LISTING 58: `lookForAlignDelims` (basic)

```
lookForAlignDelims:
   tabular: 1
   tabularx: 1
   longtable: 1
   array: 1
   matrix: 1
   ...
```

Specifying code blocks in this field instructs `latexindent.pl` to try and align each column by its alignment delimiters. It does have some limitations (discussed further in Section 10), but in many cases it will produce results such as those in Listings 59 and 60; running the command

```
cmh:~$ latexindent.pl tabular1.tex
```

gives the output given in Listing 60.

Listing 59: `tabular1.tex`

```
\begin{tabular}{cccc}
1& 2 &3       &4\\
5& &6       &\\
\end{tabular}
```

Listing 60: `tabular1.tex` default output

```
\begin{tabular}{cccc}
   1 & 2 & 3 & 4 \\
   5 &   & 6 &   \\
\end{tabular}
```

If you find that `latexindent.pl` does not perform satisfactorily on such environments then you can set the relevant key to 0, for example `tabular: 0`; alternatively, if you just want to ignore *specific* instances of the environment, you could wrap them in something from `noIndentBlock` (see Listing 43 on page 30).

If, for example, you wish to remove the alignment of the `\\` within a delimiter-aligned block, then the advanced form of `lookForAlignDelims` shown in Listing 61 is for you.

Listing 61: `lookForAlignDelims` (advanced)

```
lookForAlignDelims:
  tabular:
    delims: 1
    alignDoubleBackSlash: 1
    spacesBeforeDoubleBackSlash: 1
    multiColumnGrouping: 0
    alignRowsWithoutMaxDelims: 1
    spacesBeforeAmpersand: 1
    spacesAfterAmpersand: 1
    justification: left
    alignFinalDoubleBackSlash: 0
    dontMeasure: 0
    delimiterRegEx: (?<!\\)(&)
    delimiterJustification: left
    lookForChildCodeBlocks: 1
    alignContentAfterDoubleBackSlash: 0
    spacesAfterDoubleBackSlash: 1
  tabularx:
    delims: 1
  longtable: 1
```

Note that you can use a mixture of the basic and advanced form: in Listing 61 `tabular` and `tabularx` are advanced and `longtable` is basic. When using the advanced form, each field should receive at least 1 sub-field, and *can* (but does not have to) receive any of the following fields:

- `delims`: binary switch (0 or 1) equivalent to simply specifying, for example, `tabular: 1` in the basic version shown in Listing 58. If `delims` is set to `0` then the align at ampersand routine will not be called for this code block (default: 1);
- `alignDoubleBackSlash`: binary switch (0 or 1) to determine if `\\` should be aligned (default: 1);
- U: 2018-01-13 `spacesBeforeDoubleBackSlash`: optionally, specifies the number (integer ≥ 0) of spaces to be inserted before `\\` (default: 1);
- N: 2017-06-19 `multiColumnGrouping`: binary switch (0 or 1) that details if `latexindent.pl` should group columns above and below a `\multicolumn` command (default: 0);
- N: 2017-06-19 `alignRowsWithoutMaxDelims`: binary switch (0 or 1) that details if rows that do not contain the maximum number of delimiters should be formatted so as to have the ampersands aligned (default: 1);
- N: 2018-01-13 `spacesBeforeAmpersand`: optionally specifies the number (integer ≥ 0) of spaces to be placed *before* ampersands (default: 1);
- N: 2018-01-13 `spacesAfterAmpersand`: optionally specifies the number (integer ≥ 0) of spaces to be placed *After* ampersands (default: 1);

- N: 2018-01-13 `justification`: optionally specifies the justification of each cell as either *left* or *right* (default: left);
- N: 2020-03-21 alignFinalDoubleBackSlash optionally specifies if the *final* double backslash should be used for alignment (default: 0);
- N: 2020-03-21 dontMeasure optionally specifies if user-specified cells, rows or the largest entries should *not* be measured (default: 0);
- N: 2020-03-21 delimiterRegEx optionally specifies the pattern matching to be used for the alignment delimiter (default: `'(?<!\\)(&)'`);
- N: 2020-03-21 delimiterJustification optionally specifies the justification for the alignment delimiters (default: left); note that this feature is only useful if you have delimiters of different lengths in the same column, discussed in Section 5.5.4;
- N: 2021-12-13 lookForChildCodeBlocks optionally instructs `latexindent.pl` to search for child code blocks or not (default: 1), discussed in Section 5.5.5;
- N: 2023-05-01 alignContentAfterDoubleBackSlash optionally instructs `latexindent.pl` to align content *after* double back slash (default: 0), discussed in Section 5.5.6;
- N: 2023-05-01 spacesAfterDoubleBackSlash optionally specifies the number (integer ≥ 0) of spaces to be placed *after* the double back slash *when alignContentAfterDoubleBackSlash is active*; demonstrated in Section 5.5.6.

example 15 We will explore most of these features using the file `tabular2.tex` in Listing 62 (which contains a `\multicolumn` command), and the YAML files in Listings 63 to 69; we will explore `alignFinalDoubleBackSlash` in Listing 90; the `dontMeasure` feature will be described in Section 5.5.3, and `delimiterRegEx` in Section 5.5.4.

LISTING 62: `tabular2.tex`

```
\begin{tabular}{cccc}
A&    B & C       &D\\
AAA&    BBB & CCC       &DDD\\
  \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading}\\
one&    two & three       &four\\
five& &six      &\\
seven & \\
\end{tabular}
```

LISTING 63: `tabular2.yaml`

```
lookForAlignDelims:
   tabular:
      multiColumnGrouping: 1
```

LISTING 64: `tabular3.yaml`

```
lookForAlignDelims:
   tabular:
      alignRowsWithoutMaxDelims: 0
```

LISTING 65: `tabular4.yaml`

```
lookForAlignDelims:
   tabular:
      spacesBeforeAmpersand: 4
```

LISTING 66: `tabular5.yaml`

```
lookForAlignDelims:
   tabular:
      spacesAfterAmpersand: 4
```

LISTING 67: `tabular6.yaml`

```
lookForAlignDelims:
   tabular:
      alignDoubleBackSlash: 0
```

LISTING 68: `tabular7.yaml`

```
lookForAlignDelims:
   tabular:
      spacesBeforeDoubleBackSlash: 0
```

LISTING 69: `tabular8.yaml`

```
lookForAlignDelims:
   tabular:
      justification: "right"
```

On running the commands

```
cmh:~$ latexindent.pl tabular2.tex
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular3.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml,tabular4.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml,tabular5.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml,tabular6.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml,tabular7.yaml
cmh:~$ latexindent.pl tabular2.tex -l tabular2.yaml,tabular8.yaml
```

we obtain the respective outputs given in Listings 70 to 77.

LISTING 70: `tabular2.tex` default output

```
\begin{tabular}{cccc}
   A                                 & B                                  & C     & D    \\
   AAA                               & BBB                                & CCC   & DDD  \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading}                \\
   one                               & two                                & three & four \\
   five                              &                                    & six   &      \\
   seven                             &                                                   \\
\end{tabular}
```

LISTING 71: `tabular2.tex` using Listing 63

```
\begin{tabular}{cccc}
   A     & B                         & C     & D                          \\
   AAA   & BBB                       & CCC   & DDD                        \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
   one   & two                       & three & four                       \\
   five  &                           & six   &                            \\
   seven &                                                                \\
\end{tabular}
```

LISTING 72: `tabular2.tex` using Listing 64

```
\begin{tabular}{cccc}
   A    & B   & C     & D                                                 \\
   AAA  & BBB & CCC   & DDD                                               \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
   one  & two & three & four                                              \\
   five &     & six   &                                                   \\
   seven &                                                                \\
\end{tabular}
```

LISTING 73: `tabular2.tex` using Listings 63 and 65

```
\begin{tabular}{cccc}
   A        & B                         & C        & D                       \\
   AAA      & BBB                       & CCC      & DDD                     \\
   \multicolumn{2}{c}{first heading}    & \multicolumn{2}{c}{second heading} \\
   one      & two                       & three    & four                    \\
   five     &                           & six      &                         \\
   seven    &                                                                \\
\end{tabular}
```

LISTING 74: `tabular2.tex` using Listings 63 and 66

```
\begin{tabular}{cccc}
   A     &    B                      &    C     &    D                       \\
   AAA   &    BBB                    &    CCC   &    DDD                     \\
   \multicolumn{2}{c}{first heading} &    \multicolumn{2}{c}{second heading} \\
   one   &    two                    &    three &    four                    \\
   five  &                           &    six   &                            \\
   seven &                                                                   \\
\end{tabular}
```

LISTING 75: `tabular2.tex` using Listings 63 and 67

```
\begin{tabular}{cccc}
   A     & B                         & C     & D \\
   AAA   & BBB                       & CCC   & DDD \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
   one   & two                       & three & four \\
   five  &                           & six   & \\
   seven & \\
\end{tabular}
```

LISTING 76: `tabular2.tex` using Listings 63 and 68

```
\begin{tabular}{cccc}
   A     & B                         & C     & D                         \\
   AAA   & BBB                       & CCC   & DDD                       \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading}\\
   one   & two                       & three & four                      \\
   five  &                           & six   &                           \\
   seven &                                                               \\
\end{tabular}
```

LISTING 77: `tabular2.tex` using Listings 63 and 69

```
\begin{tabular}{cccc}
                             A &   B &                           C &    D \\
                           AAA & BBB &                         CCC &  DDD \\
   \multicolumn{2}{c}{first heading} & \multicolumn{2}{c}{second heading} \\
                           one & two &                       three & four \\
                          five &     &                         six &      \\
                         seven &                                          \\
\end{tabular}
```

Notice in particular:

- in both Listings 70 and 71 all rows have been aligned at the ampersand, even those that do not contain the maximum number of ampersands (3 ampersands, in this case);
- in Listing 70 the columns have been aligned at the ampersand;
- in Listing 71 the `\multicolumn` command has grouped the 2 columns beneath *and* above it,

because `multiColumnGrouping` is set to 1 in Listing 63;

- in Listing 72 rows 3 and 6 have *not* been aligned at the ampersand, because `alignRowsWithoutMaxDelims` has been to set to 0 in Listing 64; however, the `\\` *have* still been aligned;
- in Listing 73 the columns beneath and above the `\multicolumn` commands have been grouped (because `multiColumnGrouping` is set to 1), and there are at least 4 spaces *before* each aligned ampersand because `spacesBeforeAmpersand` is set to 4;
- in Listing 74 the columns beneath and above the `\multicolumn` commands have been grouped (because `multiColumnGrouping` is set to 1), and there are at least 4 spaces *after* each aligned ampersand because `spacesAfterAmpersand` is set to 4;
- in Listing 75 the `\\` have *not* been aligned, because `alignDoubleBackSlash` is set to `0`, otherwise the output is the same as Listing 71;
- in Listing 76 the `\\` *have* been aligned, and because `spacesBeforeDoubleBackSlash` is set to `0`, there are no spaces ahead of them; the output is otherwise the same as Listing 71;
- in Listing 77 the cells have been *right*-justified; note that cells above and below the `\multicol` statements have still been group correctly, because of the settings in Listing 63.

■

5.5.1 lookForAlignDelims: spacesBeforeAmpersand

U: 2021-06-19

The `spacesBeforeAmpersand` can be specified in a few different ways. The *basic* form is demonstrated in Listing 65, but we can customise the behaviour further by specifying if we would like this value to change if it encounters a *leading blank column*; that is, when the first column contains only zero-width entries. We refer to this as the *advanced* form.

example 16 We demonstrate this feature in relation to Listing 78; upon running the following command

```
cmh:~$ latexindent.pl aligned1.tex -o=+-default
```

then we receive the default output given in Listing 79.

LISTING 78: `aligned1.tex`

```
\begin{aligned}
& a & b, \\
& c & d.
\end{aligned}
```

LISTING 79: `aligned1-default.tex`

```
\begin{aligned}
    & a & b, \\
    & c & d.
\end{aligned}
```

The settings in Listings 80 to 83 are all equivlanent; we have used the not-yet discussed `noAdditionalIndent` field (see Section 5.8 on page 55) which will assist in the demonstration in what follows.

LISTING 80: `sba1.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned: 1
```

LISTING 81: `sba2.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand: 1
```

LISTING 82: `sba3.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand:
        default: 1
```

LISTING 83: `sba4.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand:
        leadingBlankColumn: 1
```

Upon running the following commands

■

```
cmh:~$ latexindent.pl aligned1.tex -l sba1.yaml
cmh:~$ latexindent.pl aligned1.tex -l sba2.yaml
cmh:~$ latexindent.pl aligned1.tex -l sba3.yaml
cmh:~$ latexindent.pl aligned1.tex -l sba4.yaml
```

then we receive the (same) output given in Listing 84; we note that there is *one space* before each ampersand.

LISTING 84: `aligned1-mod1.tex`

```
\begin{aligned}
 & a & b, \\
 & c & d.
\end{aligned}
```

We note in particular:

- Listing 80 demonstrates the *basic* form for `lookForAlignDelims`; in this case, the default values are specified as in Listing 61 on page 34;
- Listing 81 demonstrates the *advanced* form for `lookForAlignDelims` and specified `spacesBeforeAmpersand`. The default value is `1`;
- Listing 82 demonstrates the new *advanced* way to specify `spacesBeforeAmpersand`, and for us to set the `default` value that sets the number of spaces before ampersands which are *not* in leading blank columns. The default value is `1`.

  We note that `leadingBlankColumn` has not been specified in Listing 82, and it will inherit the value from `default`;
- Listing 83 demonstrates spaces to be used before amperands for *leading blank columns*. We note that *default* has not been specified, and it will be set to `1` by default.

■

example 17 We can customise the space before the ampersand in the *leading blank column* of Listing 84 by using either of Listings 85 and 86, which are equivalent.

LISTING 85: `sba5.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand:
        leadingBlankColumn: 0
```

LISTING 86: `sba6.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand:
        leadingBlankColumn: 0
        default: 1
```

Upon running

```
cmh:~$ latexindent.pl aligned1.tex -l sba5.yaml
cmh:~$ latexindent.pl aligned1.tex -l sba6.yaml
```

then we receive the (same) output given in Listing 87. We note that the space before the ampersand in the *leading blank column* has been set to `0` by Listing 86.

We can demonstrated this feature further using the settings in Listing 89 which give the output in Listing 88.

■

Listing 87: `aligned1-mod5.tex`

```
\begin{aligned}
& a & b, \\
& c & d.
\end{aligned}
```

Listing 88: `aligned1.tex` using Listing 89

```
\begin{aligned}
   & a& b, \\
   & c& d.
\end{aligned}
```

Listing 89: `sba7.yaml`

```
noAdditionalIndent:
  aligned: 1
lookForAlignDelims:
   aligned:
      spacesBeforeAmpersand:
        leadingBlankColumn: 3
        default: 0
```

5.5.2 lookForAlignDelims: alignFinalDoubleBackSlash

There may be times when a line of a code block contains more than `\\`, and in which case, you may want the *final* double backslash to be aligned.

example 18 We explore the `alignFinalDoubleBackSlash` feature by using the file in Listing 90. Upon running the following commands

N: 2020-03-21

```
cmh:~$ latexindent.pl tabular4.tex -o=+-default
cmh:~$ latexindent.pl tabular4.tex -o=+-FDBS
    -y="lookForAlignDelims:tabular:alignFinalDoubleBackSlash:1"
```

then we receive the respective outputs given in Listing 91 and Listing 92.

Listing 90: `tabular4.tex`

```
\begin{tabular}{lc}
    Name & \shortstack{Hi \\ Lo} \\
    Foo  & Bar            \\
\end{tabular}
```

Listing 91: `tabular4-default.tex`

```
\begin{tabular}{lc}
    Name & \shortstack{Hi \\ Lo} \\
    Foo  & Bar            \\
\end{tabular}
```

Listing 92: `tabular4-FDBS.tex`

```
\begin{tabular}{lc}
    Name & \shortstack{Hi \\ Lo} \\
    Foo  & Bar                   \\
\end{tabular}
```

We note that in:

- Listing 91, by default, the *first* set of double back slashes in the first row of the `tabular` environment have been used for alignment;
- Listing 92, the *final* set of double backslashes in the first row have been used, because we specified `alignFinalDoubleBackSlash` as 1.

As of Version 3.0, the alignment routine works on mandatory and optional arguments within commands, and also within 'special' code blocks (see `specialBeginEnd` on page 47).

example 19 Assuming that you have a command called `\matrix` and that it is populated within `lookForAlignDelims` (which it is, by default), and that you run the command

```
cmh:~$ latexindent.pl matrix1.tex
```

then the before-and-after results shown in Listings 93 and 94 are achievable by default.

Listing 93: `matrix1.tex`

```
\matrix [
   1&2   &3\\
4&5&6]{
7&8   &9\\
10&11&12
}
```

Listing 94: `matrix1.tex` default output

```
\matrix [
   1 & 2 & 3 \\
   4 & 5 & 6]{
   7  & 8  & 9  \\
   10 & 11 & 12
}
```

If you have blocks of code that you wish to align at the & character that are *not* wrapped in, for example, `\begin{tabular}` … `\end{tabular}`, then you can use the mark up illustrated in Listing 95; the default output is shown in Listing 96. Note that the `%*` must be next to each other, but that there can be any number of spaces (possibly none) between the `*` and `\begin{tabular}`; note also that you may use any environment name that you have specified in `lookForAlignDelims`.

Listing 95: `align-block.tex`

```
%* \begin{tabular}
   1 & 2 & 3 & 4 \\
   5 &   & 6 &   \\
  %* \end{tabular}
```

Listing 96: `align-block.tex` default output

```
%* \begin{tabular}
   1 & 2 & 3 & 4 \\
   5 &   & 6 &   \\
%* \end{tabular}
```

With reference to Table 2 on page 56 and the, yet undiscussed, fields of `noAdditionalIndent` and `indentRules` (see Section 5.8 on page 55), these comment-marked blocks are considered `environments`.

5.5.3 lookForAlignDelims: the dontMeasure feature

N: 2020-03-21

The `lookForAlignDelims` field can, optionally, receive the `dontMeasure` option which can be specified in a few different ways.

example 20 We will explore this feature in relation to the code given in Listing 97; the default output is shown in Listing 98.

Listing 97: `tabular-DM.tex`

```
\begin{tabular}{cccc}
  aaaaaa&bbbbb&ccc&dd\\
  11&2&33&4\\
  5&66&7&8
\end{tabular}
```

Listing 98: `tabular-DM.tex` default output

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11     & 2     & 33  & 4  \\
   5      & 66    & 7   & 8
\end{tabular}
```

The `dontMeasure` field can be specified as `largest`, and in which case, the largest element will not be measured; with reference to the YAML file given in Listing 100, we can run the command

```
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure1.yaml
```

and receive the output given in Listing 99.

Listing 99: `tabular-DM.tex` using Listing 100

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11 & 2  & 33 & 4          \\
   5  & 66 & 7  & 8
\end{tabular}
```

Listing 100: `dontMeasure1.yaml`

```
lookForAlignDelims:
   tabular:
      dontMeasure: largest
```

We note that the *largest* column entries have not contributed to the measuring routine. ■

example 21 The `dontMeasure` field can also be specified in the form demonstrated in Listing 102. On running the following commands,

```
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure2.yaml
```

we receive the output in Listing 101. ■

LISTING 101: `tabular-DM.tex` using Listing 102 or Listing 104

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11 & 2  & 33 & 4          \\
   5  & 66 & 7  & 8
\end{tabular}
```

LISTING 102: `dontMeasure2.yaml`

```
lookForAlignDelims:
   tabular:
      dontMeasure:
        - aaaaaa
        - bbbbb
        - ccc
        - dd
```

We note that in Listing 102 we have specified entries not to be measured, one entry per line. ■

example 22 The `dontMeasure` field can also be specified in the forms demonstrated in Listing 104 and Listing 105. Upon running the commands

```
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure3.yaml
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure4.yaml
```

we receive the output given in Listing 103

LISTING 103: `tabular-DM.tex` using Listing 104 or Listing 104

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11 & 2  & 33 & 4          \\
   5  & 66 & 7  & 8
\end{tabular}
```

LISTING 104: `dontMeasure3.yaml`

```
lookForAlignDelims:
   tabular:
      dontMeasure:
        -
          this: aaaaaa
          applyTo: cell
        -
          this: bbbbb
        - ccc
        - dd
```

LISTING 105: `dontMeasure4.yaml`

```
lookForAlignDelims:
   tabular:
      dontMeasure:
        -
          regex: [a-z]
          applyTo: cell
```

We note that in:

- Listing 104 we have specified entries not to be measured, each one has a *string* in the `this` field, together with an optional specification of `applyTo` as `cell`;
- Listing 105 we have specified entries not to be measured as a *regular expression* using the `regex` field, together with an optional specification of `applyTo` as `cell` field, together with an optional specification of `applyTo` as `cell`.

In both cases, the default value of `applyTo` is `cell`, and does not need to be specified. ■

example 23 We may also specify the `applyTo` field as `row`, a demonstration of which is given in Listing 107; upon running

```
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure5.yaml
```

we receive the output in Listing 106.

LISTING 106: `tabular-DM.tex` using Listing 107

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11 & 2  & 33 & 4          \\
   5  & 66 & 7  & 8
\end{tabular}
```

LISTING 107: `dontMeasure5.yaml`

```
lookForAlignDelims:
   tabular:
      dontMeasure:
        -
          this: aaaaaa&bbbbb&ccc&dd\\
          applyTo: row
```

■

example 24 Finally, the `applyTo` field can be specified as `row`, together with a `regex` expression. For example, for the settings given in Listing 109, upon running

```
cmh:~$ latexindent.pl tabular-DM.tex -l=dontMeasure6.yaml
```

we receive the output in Listing 108.

LISTING 108: `tabular-DM.tex` using Listing 109

```
\begin{tabular}{cccc}
   aaaaaa & bbbbb & ccc & dd \\
   11 & 2  & 33 & 4          \\
   5  & 66 & 7  & 8
\end{tabular}
```

LISTING 109: `dontMeasure6.yaml`

```
lookForAlignDelims:
   tabular:
      dontMeasure:
        -
          regex: [a-z]
          applyTo: row
```

■

5.5.4 lookForAlignDelims: the delimiterRegEx and delimiterJustification feature

N: 2020-03-21

The delimiter alignment will, by default, align code blocks at the ampersand character. The behaviour is controlled by the `delimiterRegEx` field within `lookForAlignDelims`; the default value is `'(?<!\\)(&)'`, which can be read as: *an ampersand, as long as it is not immediately preceded by a backslash*.

Warning!

Important: note the ‘capturing’ parenthesis in the `(&)` which are necessary; if you intend to customise this field, then be sure to include them appropriately.

example 25 We demonstrate how to customise this with respect to the code given in Listing 110; the default output from `latexindent.pl` is given in Listing 111.

LISTING 110: `tabbing.tex`

```
\begin{tabbing}
    aa \=   bb \= cc \= dd \= ee \\
    \>2\> 1 \> 7 \> 3 \\
    \>3 \> 2\>8\> 3 \\
    \>4 \>2 \\
\end{tabbing}
```

LISTING 111: `tabbing.tex` default output

```
\begin{tabbing}
   aa \=   bb \= cc \= dd \= ee \\
   \>2\> 1 \> 7 \> 3 \\
   \>3 \> 2\>8\> 3 \\
   \>4 \>2 \\
\end{tabbing}
```

Let’s say that we wish to align the code at either the `\=` or `\>`. We employ the settings given in Listing 113 and run the command

```
cmh:~$ latexindent.pl tabbing.tex -l=delimiterRegEx1.yaml
```

to receive the output given in Listing 112.

LISTING 112: `tabbing.tex` using Listing 113

```
\begin{tabbing}
   aa \= bb \= cc \= dd \= ee \\
      \> 2  \> 1  \> 7  \> 3  \\
      \> 3  \> 2  \> 8  \> 3  \\
      \> 4  \> 2              \\
\end{tabbing}
```

LISTING 113: `delimiterRegEx1.yaml`

```
lookForAlignDelims:
   tabbing:
    delimiterRegEx: '(\\(?:=|>))'
```

We note that:

- in Listing 112 the code has been aligned, as intended, at both the `\=` and `\>`;

■

- in Listing 113 we have heeded the warning and captured the expression using grouping parenthesis, specified a backslash using `\\` and said that it must be followed by either `=` or `>`.

■

example 26 We can explore `delimiterRegEx` a little further using the settings in Listing 115 and run the command

```
cmh:~$ latexindent.pl tabbing.tex -l=delimiterRegEx2.yaml
```

to receive the output given in Listing 114.

LISTING 114: `tabbing.tex` using Listing 115

```
\begin{tabbing}
   aa \=   bb \= cc \= dd \= ee \\
    \> 2 \> 1 \> 7 \> 3         \\
    \> 3 \> 2 \> 8 \> 3         \\
    \> 4 \> 2                   \\
\end{tabbing}
```

LISTING 115: `delimiterRegEx2.yaml`

```
lookForAlignDelims:
   tabbing:
    delimiterRegEx: '(\\>)'
```

We note that only the `\>` have been aligned.

■

example 27 Of course, the other lookForAlignDelims options can be used alongside the `delimiterRegEx`; regardless of the type of delimiter being used (ampersand or anything else), the fields from Listing 61 on page 34 remain the same; for example, using the settings in Listing 117, and running

```
cmh:~$ latexindent.pl tabbing.tex -l=delimiterRegEx3.yaml
```

to receive the output given in Listing 116.

LISTING 116: `tabbing.tex` using Listing 117

```
\begin{tabbing}
   aa\=bb\=cc\=dd\=ee \\
     \>2 \>1 \>7 \>3  \\
     \>3 \>2 \>8 \>3  \\
     \>4 \>2          \\
\end{tabbing}
```

LISTING 117: `delimiterRegEx3.yaml`

```
lookForAlignDelims:
   tabbing:
    delimiterRegEx: '(\\(?:=|>))'
    spacesBeforeAmpersand: 0
    spacesAfterAmpersand: 0
```

■

example 28 It is possible that delimiters specified within `delimiterRegEx` can be of different lengths. Consider the file in Listing 118, and associated YAML in Listing 120. Note that the Listing 120 specifies the option for the delimiter to be either `#` or `\>`, *which are different lengths*. Upon running the command

```
cmh:~$ latexindent.pl tabbing1.tex -l=delimiterRegEx4.yaml -o=+-mod4
```

we receive the output in Listing 119.

LISTING 118: `tabbing1.tex`

```
\begin{tabbing}
    1#22\>333\\
    xxx#aaa#yyyyy\\
    .##&\\
\end{tabbing}
```

LISTING 119: `tabbing1-mod4.tex`

```
\begin{tabbing}
   1   # 22  \> 333   \\
   xxx # aaa #  yyyyy \\
   .   #     #  &     \\
\end{tabbing}
```

LISTING 120: `delimiterRegEx4.yaml`

```
lookForAlignDelims:
   tabbing:
    delimiterRegEx: '(#|\\>)'
```

■

example 29 You can set the *delimiter* justification as either `left` (default) or `right`, which will only have effect when delimiters in the same column have different lengths. Using the settings in Listing 122 and running the command

```
cmh:~$ latexindent.pl tabbing1.tex -l=delimiterRegEx5.yaml -o=+-mod5
```

gives the output in Listing 121.

LISTING 121: `tabbing1-mod5.tex`

```
\begin{tabbing}
   1   # 22  \> 333   \\
   xxx # aaa  # yyyyy \\
   .   #      # &     \\
\end{tabbing}
```

LISTING 122: `delimiterRegEx5.yaml`

```
lookForAlignDelims:
   tabbing:
    delimiterRegEx: '(#|\\>)'
    delimiterJustification: right
```

Note that in Listing 121 the second set of delimiters have been *right aligned* – it is quite subtle! ■

5.5.5 lookForAlignDelims: lookForChildCodeBlocks

N: 2021-12-13

There may be scenarios in which you would prefer to instruct `latexindent.pl` *not* to search for child blocks; in which case setting `lookForChildCodeBlocks` to 0 may be a good way to proceed.

example 30 Using the settings from Listing 100 on page 41 on the file in Listing 123 and running the command

```
cmh:~$ latexindent.pl tabular-DM-1.tex -l=dontMeasure1.yaml -o=+-mod1
```

gives the output in Listing 124.

LISTING 123: `tabular-DM-1.tex`

```
\begin{tabular}{cc}
1&2\only<2->{\\
3&4}
\end{tabular}
```

LISTING 124: `tabular-DM-1-mod1.tex`

```
\begin{tabular}{cc}
   1 & 2\only<2->{ \\
      3 & 4}
\end{tabular}
```

We can improve the output from Listing 124 by employing the settings in Listing 126

```
cmh:~$ latexindent.pl tabular-DM-1.tex -l=dontMeasure1a.yaml -o=+-mod1a
```

which gives the output in Listing 126.

LISTING 125: `tabular-DM-1-mod1a.tex`

```
\begin{tabular}{cc}
   1 & 2\only<2->{ \\
   3 & 4}
\end{tabular}
```

LISTING 126: `dontMeasure1a.yaml`

```
lookForAlignDelims:
   tabular:
      dontMeasure: largest
      lookForChildCodeBlocks: 0
```

■

5.5.6 lookForAlignDelims: alignContentAfterDoubleBackSlash

N: 2023-05-01

You can instruct `latexindent` to align content after the double back slash. See also Section 6.3.2 on page 118.

example 31 We consider the file in Listing 127, and the default output given in Listing 128. ■

LISTING 127: `tabular5.tex`

```
\begin{tabular}{cc}
  1 & 2
  \\ aa & bbb
  \\ ccc&ddd
\end{tabular}
```

LISTING 128: `tabular5-default.tex`

```
\begin{tabular}{cc}
   1 & 2
   \\ aa & bbb
   \\ ccc&ddd
\end{tabular}
```

Using the settings given in Listing 130 and running

```
cmh:~$ latexindent.pl -s tabular5.tex -l alignContentAfterDBS1 -o=+-mod1
```

gives the output in Listing 129.

LISTING 129: `tabular5-mod1.tex`

```
\begin{tabular}{cc}
      1   & 2
   \\ aa  & bbb
   \\ ccc & ddd
\end{tabular}
```

LISTING 130: `alignContentAfterDBS1.yaml`

```
lookForAlignDelims:
   tabular:
      alignContentAfterDoubleBackSlash: 1
```

example 32 When using the `alignContentAfterDoubleBackSlash` feature, then you can also specify how many spaces to insert after the double backslash; the default is 1.

N: 2023-05-01

Starting from Listing 127 and using the the settings given in Listing 132

```
cmh:~$ latexindent.pl -s tabular5.tex -l alignContentAfterDBS2 -o=+-mod2
```

gives the output in Listing 131.

LISTING 131: `tabular5-mod2.tex`

```
\begin{tabular}{cc}
        1   & 2
   \\   aa  & bbb
   \\   ccc & ddd
\end{tabular}
```

LISTING 132: `alignContentAfterDBS2.yaml`

```
lookForAlignDelims:
   tabular:
      alignContentAfterDoubleBackSlash: 1
      spacesAfterDoubleBackSlash: 3
```

5.6 Indent after items, specials and headings

`indentAfterItems`: ⟨*fields*⟩

The environment names specified in `indentAfterItems` tell `latexindent.pl` to look for `\item` commands; if these switches are set to 1 then indentation will be performed so as indent the code after each `item`. A demonstration is given in Listings 134 and 135

LISTING 133: `indentAfterItems`

```
indentAfterItems:
  itemize: 1
  itemize*: 1
  enumerate: 1
  enumerate*: 1
  description: 1
  description*: 1
  list: 1
```

LISTING 134: `items1.tex`

```
\begin{itemize}
\item some text here
some more text here
some more text here
\item another item
some more text here
\end{itemize}
```

LISTING 135: `items1.tex` default output

```
\begin{itemize}
   \item some text here
         some more text here
         some more text here
   \item another item
         some more text here
\end{itemize}
```

`itemNames`: ⟨*fields*⟩

If you have your own `item` commands (perhaps you prefer to use `myitem`, for example) then you can put populate them in `itemNames`. For example, users of the `exam` document class might like to add `parts` to `indentAfterItems` and `part` to `itemNames` to their user settings (see Section 4 on page 23 for details of how to configure user settings, and Listing 33 on page 24
in particular .)

LISTING 136: `itemNames`

```
itemNames:
  item: 1
  myitem: 1
```

`specialBeginEnd`: ⟨*fields*⟩

U: 2017-08-21

The fields specified in `specialBeginEnd` are, in their default state, focused on math mode begin and end statements, but there is no requirement for this to be the case; Listing 137 shows the default settings of `specialBeginEnd`.

LISTING 137: `specialBeginEnd`

```
specialBeginEnd:
  displayMath:
    begin: (?<!\\)\\\[            # \[ but *not* \\[
    end: \\\]                     # \]
    lookForThis: 1
  inlineMath:
    begin: (?<!\$)(?<!\\)\$(?!\$) # $ but *not* \$ or $$
    body: [^$]*?                  # anything *except* $
    end: (?<!\\)\$(?!\$)          # $ but *not* \$ or $$
    lookForThis: 1
  displayMathTeX:
    begin: \$\$                   # $$
    end: \$\$                     # $$
    lookForThis: 1
  specialBeforeCommand: 0
```

The field `displayMath` represents `\[...\]`, `inlineMath` represents `$...$` and `displayMathTex` represents `$$...$$`. You can, of course, rename these in your own YAML files (see Section 4.2 on page 24); indeed, you might like to set up your own special begin and end statements.

example 33 A demonstration of the before-and-after results are shown in Listings 138 and 139; explicitly, running the command

```
cmh:~$ latexindent.pl special1.tex -o=+-default
```

gives the output given in Listing 139.

■

LISTING 138: `special1.tex` before

```
The function $f$ has formula
\[
f(x)=x^2.
\]
If you like splitting dollars,
$
g(x)=f(2x)
$
```

LISTING 139: `special1.tex` default output

```
The function $f$ has formula
\[
   f(x)=x^2.
\]
If you like splitting dollars,
$
   g(x)=f(2x)
$
```

For each field, `lookForThis` is set to 1 by default, which means that `latexindent.pl` will look for this pattern; you can tell `latexindent.pl` not to look for the pattern, by setting `lookForThis` to 0.

N: 2017-08-21

There are examples in which it is advantageous to search for `specialBeginEnd` fields *before* searching for commands, and the `specialBeforeCommand` switch controls this behaviour.

example 34 For example, consider the file shown in Listing 140.

LISTING 140: `specialLR.tex`

```
\begin{equation}
\left[
\sqrt{
a+b
}
\right]
\end{equation}
```

Now consider the YAML files shown in Listings 141 and 142

LISTING 141: `specialsLeftRight.yaml`

```
specialBeginEnd:
    leftRightSquare:
        begin: '\\left\['
        end: '\\right\]'
        lookForThis: 1
```

LISTING 142: `specialBeforeCommand.yaml`

```
specialBeginEnd:
    specialBeforeCommand: 1
```

Upon running the following commands

```
cmh:~$ latexindent.pl specialLR.tex -l=specialsLeftRight.yaml
cmh:~$ latexindent.pl specialLR.tex -l=specialsLeftRight.yaml,specialBeforeCommand.yaml
```

we receive the respective outputs in Listings 143 and 144.

LISTING 143: `specialLR.tex` using Listing 141

```
\begin{equation}
   \left[
      \sqrt{
         a+b
      }
      \right]
\end{equation}
```

LISTING 144: `specialLR.tex` using Listings 141 and 142

```
\begin{equation}
   \left[
      \sqrt{
         a+b
      }
   \right]
\end{equation}
```

Notice that in:

- Listing 143 the `\left` has been treated as a *command*, with one optional argument;
- Listing 144 the `specialBeginEnd` pattern in Listing 141 has been obeyed because List-

ing 142 specifies that the `specialBeginEnd` should be sought *before* commands. ■

N: 2018-04-27 You can, optionally, specify the `middle` field for anything that you specify in `specialBeginEnd`.

example 35 For example, let's consider the `.tex` file in Listing 145.

LISTING 145: `special2.tex`

```
\If
something 0
\ElsIf
something 1
\ElsIf
something 2
\ElsIf
something 3
\Else
something 4
\EndIf
```

Upon saving the YAML settings in Listings 147 and 149 and running the commands

```
cmh:~$ latexindent.pl special2.tex -l=middle
cmh:~$ latexindent.pl special2.tex -l=middle1
```

then we obtain the output given in Listings 146 and 148.

LISTING 146: `special2.tex` using Listing 147

```
\If
   something 0
\ElsIf
   something 1
\ElsIf
   something 2
\ElsIf
   something 3
   \Else
   something 4
\EndIf
```

LISTING 147: `middle.yaml`

```
specialBeginEnd:
    If:
        begin: '\\If'
        middle: '\\ElsIf'
        end: '\\EndIf'
        lookForThis: 1
```

LISTING 148: `special2.tex` using Listing 149

```
\If
   something 0
\ElsIf
   something 1
\ElsIf
   something 2
\ElsIf
   something 3
\Else
   something 4
\EndIf
```

LISTING 149: `middle1.yaml`

```
specialBeginEnd:
    If:
        begin: '\\If'
        middle:
          - '\\ElsIf'
          - '\\Else'
        end: '\\EndIf'
        lookForThis: 1
```

We note that:

- in Listing 146 the bodies of each of the `Elsif` statements have been indented appropriately;
- the `Else` statement has *not* been indented appropriately in Listing 146 – read on! ■

- we have specified multiple settings for the `middle` field using the syntax demonstrated in Listing 149 so that the body of the `Else` statement has been indented appropriately in Listing 148. ■

You may need these fields in your own YAML files (see Section 4.2 on page 24), if you use popular algorithm packages such as algorithms, algorithm2e or algpseudocode, etc.

example 36 For example, let's consider the `.tex` file in Listing 150.

LISTING 150: `specialAlgo.tex`

```
\For{$n = 1, \dots, 10$}
\State body
\EndFor
\FOR{for 1}
\FOR{for 2}
\FOR{for 3}
\STATE{some statement.}
\ENDFOR
\ENDFOR
\ENDFOR
\If{$quality\ge 9$}
\State $a\gets perfect$
\ElsIf{$quality\ge 7$}
\State $a\gets good$
\Else
\While{$i\le n$}
\State $i\gets i+1$
\EndWhile
\EndIf
\ForAll{$n \in \{1, \dots, 10\}$}
\State body
\Loop
\State body
\EndLoop
\State $i\gets 1$
\Repeat
\State $i\gets i+1$
\Until{$i>n$}
\EndFor
\Function{Euclid}{$a,b$}\Comment{The g.c.d. of a and b}
\While{$r\not=0$}\Comment{We have the answer if r is 0}
\State $r\gets a\bmod b$
\EndWhile
\State \textbf{return} $b$\Comment{The gcd is b}
\EndFunction
```

Upon saving the YAML settings in Listing 152 and running the command

```
cmh:~$ latexindent.pl -l=algo.yaml specialAlgo.tex
```

then we obtain the output given in Listing 151.

LISTING 151: `specialAlgo.tex` using Listing 152

```
\For{$n = 1, \dots, 10$}
    \State body
\EndFor
\FOR{for 1}
    \FOR{for 2}
        \FOR{for 3}
            \STATE{some statement.}
        \ENDFOR
    \ENDFOR
\ENDFOR
\If{$quality\ge 9$}
    \State $a\gets perfect$
\ElsIf{$quality\ge 7$}
    \State $a\gets good$
\Else
    \While{$i\le n$}
        \State $i\gets i+1$
    \EndWhile
\EndIf
\ForAll{$n \in \{1, \dots, 10\}$}
    \State body
    \Loop
        \State body
    \EndLoop
    \State $i\gets 1$
    \Repeat
        \State $i\gets i+1$
    \Until{$i>n$}
\EndFor
\Function{Euclid}{$a,b$}\Comment{The g.c.d. of a and b}
    \While{$r\not=0$}\Comment{We have the answer if r is 0}
        \State $r\gets a\bmod b$
    \EndWhile
    \State \textbf{return} $b$\Comment{The gcd is b}
\EndFunction
```

LISTING 152: `algo.yaml`

```
specialBeginEnd:
  ForStatement:
    begin: \\For\{[^}]+?\}
    end: \\EndFor
  FORStatement:
    begin: \\FOR\{[^}]+?\}
    end: \\ENDFOR
  WhileStatement:
    begin: \\While\{[^}]+?\}
    end: \\EndWhile
  WHILEStatement:
    begin: \\WHILE\{[^}]+?\}
    end: \\ENDWHILE
  ForAllStatement:
    begin: \\ForAll\{[^}]+?\}
    end: \\EndFor
  LoopStatement:
    begin: \\Loop
    end: \\EndLoop
  RepeatStatement:
    begin: \\Repeat
    end: \\Until\{[^}]+?\}
  ProcedureStatement:
    begin: \\Procedure\{[^}]+?\}\{[^}]+?\}
    end: \\EndProcedure
  FunctionStatement:
    begin: \\Function\{[^}]+?\}\{[^}]+?\}
    end: \\EndFunction
  IfStatement:
    begin: \\If\{[^}]+?\}
    middle:
      - \\Else
      - \\ElsIf\{[^}]+?\}
    end: \\EndIf
  IFStatement:
    begin: \\IF\{[^}]+?\}
    middle:
      - \\ELSE
      - \\ELSIF\{[^}]+?\}
    end: \\ENDIF
  specialBeforeCommand: 1
```

■

N: 2018-08-13

You may specify fields in `specialBeginEnd` to be treated as verbatim code blocks by changing `lookForThis` to be `verbatim`.

example 37 For example, beginning with the code in Listing 153 and the YAML in Listing 154, and running

```
cmh:~$ latexindent.pl special3.tex -l=special-verb1
```

then the output in Listing 153 is unchanged.

Listing 153: `special3.tex` and output using Listing 154

```
\[
  special code
blocks
    can be
  treated
    as verbatim\]
```

Listing 154: `special-verb1.yaml`

```
specialBeginEnd:
    displayMath:
        lookForThis: verbatim
```

We can combine the `specialBeginEnd` with the `lookForAlignDelims` feature.

example 38 We begin with the code in Listing 155.

Listing 155: `special-align.tex`

```
\begin{tikzpicture}
  \path (A) edge node {0,1,L}(B)
  edge node {1,1,R} (C)
  (B) edge [loop above]node {1,1,L}(B)
  edge node {0,1,L}(C)
  (C) edge node {0,1,L}(D)
  edge [bend left]node {1,0,R}(E)
  (D) edge[loop below] node {1,1,R}(D)
  edge node {0,1,R}(A)
  (E) edge[bend left] node {1,0,R} (A);
\end{tikzpicture}
```

Let's assume that our goal is to align the code at the `edge` and `node` text; we employ the code given in Listing 157 and run the command

```
cmh:~$ latexindent.pl special-align.tex -l edge-node1.yaml -o=+-mod1
```

to receive the output in Listing 156.

Listing 156: `special-align.tex` using Listing 157

```
\begin{tikzpicture}
   \path (A) edge              node {0,1,L}(B)
             edge              node {1,1,R} (C)
         (B) edge [loop above] node {1,1,L}(B)
             edge              node {0,1,L}(C)
         (C) edge              node {0,1,L}(D)
             edge [bend left]  node {1,0,R}(E)
         (D) edge [loop below] node {1,1,R}(D)
             edge              node {0,1,R}(A)
         (E) edge [bend left]  node {1,0,R} (A);
\end{tikzpicture}
```

Listing 157: `edge-node1.yaml`

```
specialBeginEnd:
    path:
        begin: '\\path'
        end: ';'
        lookForThis: 1
    specialBeforeCommand: 1

lookForAlignDelims:
   path:
      delimiterRegEx: '(edge|node)'
```

The output in Listing 156 is not quite ideal. We can tweak the settings within Listing 157 in order to improve the output; in particular, we employ the code in Listing 159 and run the command

```
cmh:~$ latexindent.pl special-align.tex -l edge-node2.yaml -o=+-mod2
```

to receive the output in Listing 158.

LISTING 158: `special-align.tex` using Listing 159

```
\begin{tikzpicture}
   \path (A) edge                 node {0,1,L} (B)
             edge                 node {1,1,R} (C)
         (B) edge [loop above] node {1,1,L} (B)
             edge                 node {0,1,L} (C)
         (C) edge                 node {0,1,L} (D)
             edge [bend left]  node {1,0,R} (E)
         (D) edge [loop below] node {1,1,R} (D)
             edge                 node {0,1,R} (A)
         (E) edge [bend left]  node {1,0,R} (A);
\end{tikzpicture}
```

LISTING 159: `edge-node2.yaml`

```
specialBeginEnd:
    path:
        begin: '\\path'
        end: ';'
    specialBeforeCommand: 1

lookForAlignDelims:
   path:
      delimiterRegEx:
   '(edge|node\h*\{[0-9,A-Z]+\})'
```

■

U: 2021-06-19 The `lookForThis` field can be considered optional; by default, it is assumed to be 1, which is demonstrated in Listing 159.

N: 2023-09-23 Referencing Listing 137 on page 47 we see that each of the `specialBeginEnd` fields can *optionally* accept the `body` field. If the `body` field is omitted, then `latexindent.pl` uses a value that means

> anything except one of the begin statements from `specialBeginEnd`.

In general, it is usually *not* necessary to specify the `body` field, but let's detail an example just for reference.

example 39 We begin with the example in Listing 160

LISTING 160: `special-body.tex`

```
$
a
+
(
b + c
-
(
    d
)
)
=
e
$
and
$
f + g = h
$
```

Using the settings in Listing 162 and running the command

```
cmh:~$ latexindent.pl special-body.tex -l=special-body1.yaml
```

gives the output in Listing 161.

■

LISTING 161: `special-body.tex` using Listing 162

```
$
  a
  +
  (
    b + c
    -
    (
      d
    )
  )
  =
  e
$
and
$
  f + g = h
$
```

LISTING 162: `special-body1.yaml`

```
defaultIndent: "  "
specialBeginEnd:
  specialBeforeCommand: 1
  parentheses:
    begin: \(
    end: \)
```

We note that the output in Listing 161 is as we would expect, even *without* the `body` field specified.

Another option (purely for reference) that leaves the output in Listing 161 unchanged is shown in Listing 163.

LISTING 163: `special-body2.yaml`

```
defaultIndent: "  "
specialBeginEnd:
  specialBeforeCommand: 1
  parentheses:
    begin: \(
    body: [^()]*?
    end: \)
```

The `body` field in Listing 163 means *anything except (or)*.

■

`indentAfterHeadings`: ⟨*fields*⟩

This field enables the user to specify indentation rules that take effect after heading commands such as `\part`, `\chapter`, `\section`, `\subsection*`, or indeed any user-specified command written in this field.[5]

LISTING 164: `indentAfterHeadings`

```
indentAfterHeadings:
  part:
    indentAfterThisHeading: 0
    level: 1
  chapter:
    indentAfterThisHeading: 0
    level: 2
  section:
    indentAfterThisHeading: 0
    level: 3
```

The default settings do *not* place indentation after a heading, but you can easily switch them on by changing `indentAfterThisHeading` from 0 to 1. The `level` field tells `latexindent.pl` the hierarchy of the heading structure in your document. You might, for example, like to have both `section` and `subsection` set with `level: 3` because you do not want the indentation to go too

---

[5] There is a slight difference in interface for this field when comparing Version 2.2 to Version 3.0; see Section M on page 180 for details.

deep.

You can add any of your own custom heading commands to this field, specifying the `level` as appropriate. You can also specify your own indentation in `indentRules` (see Section 5.8 on the following page); you will find the default `indentRules` contains `chapter: " "` which tells `latexindent.pl` simply to use a space character after `chapter` headings (once `indent` is set to `1` for `chapter`).

example 40 For example, assuming that you have the code in Listing 166 saved into `headings1.yaml`, and that you have the text from Listing 165 saved into `headings1.tex`.

Listing 165: `headings1.tex`

```
\subsection{subsection title}
subsection text
subsection text
\paragraph{paragraph title}
paragraph text
paragraph text
\paragraph{paragraph title}
paragraph text
paragraph text
```

Listing 166: `headings1.yaml`

```
indentAfterHeadings:
    subsection:
       indentAfterThisHeading: 1
       level: 1
    paragraph:
       indentAfterThisHeading: 1
       level: 2
```

If you run the command

```
cmh:~$ latexindent.pl headings1.tex -l=headings1.yaml
```

then you should receive the output given in Listing 167.

Listing 167: `headings1.tex` using Listing 166

```
\subsection{subsection title}
␣subsection text
␣subsection text
␣\paragraph{paragraph title}
␣␣paragraph text
␣␣paragraph text
␣\paragraph{paragraph title}
␣␣paragraph text
␣␣paragraph text
```

Listing 168: `headings1.tex` second modification

```
\subsection{subsection title}
␣subsection text
␣subsection text
\paragraph{paragraph title}
␣paragraph text
␣paragraph text
\paragraph{paragraph title}
␣paragraph text
␣paragraph text
```

■

Now say that you modify the `YAML` from Listing 166 so that the `paragraph` `level` is `1`; after running

```
cmh:~$ latexindent.pl headings1.tex -l=headings1.yaml
```

you should receive the code given in Listing 168; notice that the `paragraph` and `subsection` are at the same indentation level.

`maximumIndentation`: ⟨*horizontal space*⟩

N: 2017-08-21

You can control the maximum indentation given to your file by specifying the `maximumIndentation` field as horizontal space (but *not* including tabs). This feature uses the `Text::Tabs` module [47], and is *off* by default.

example 41 For example, consider the example shown in Listing 169 together with the default output shown in Listing 170.

■

LISTING 169: `mult-nested.tex`

```
\begin{one}
one
\begin{two}
    two
\begin{three}
     three
\begin{four}
       four
\end{four}
\end{three}
\end{two}
\end{one}
```

LISTING 170: `mult-nested.tex` default output

```
\begin{one}
⎵one
⎵\begin{two}
⎵⎵two
⎵⎵\begin{three}
⎵⎵⎵three
⎵⎵⎵\begin{four}
⎵⎵⎵⎵four
⎵⎵⎵\end{four}
⎵⎵\end{three}
⎵\end{two}
\end{one}
```

■

example 42 Now say that, for example, you have the `max-indentation1.yaml` from Listing 172 and that you run the following command:

```
cmh:~$ latexindent.pl mult-nested.tex -l=max-indentation1
```

You should receive the output shown in Listing 171.

LISTING 171: `mult-nested.tex` using Listing 172

```
\begin{one}
␣one
␣\begin{two}
␣two
␣\begin{three}
␣three
␣\begin{four}
␣four
␣\end{four}
␣\end{three}
␣\end{two}
\end{one}
```

LISTING 172: `max-indentation1.yaml`

```
maximumIndentation: " "
```

■

Comparing the output in Listings 170 and 171 we notice that the (default) tabs of indentation have been replaced by a single space.

In general, when using the `maximumIndentation` feature, any leading tabs will be replaced by equivalent spaces except, of course, those found in `verbatimEnvironments` (see Listing 37 on page 29) or `noIndentBlock` (see Listing 43 on page 30).

5.7 The code blocks known latexindent.pl

As of Version 3.0, `latexindent.pl` processes documents using code blocks; each of these are shown in Table 2.

N: 2019-07-13

We will refer to these code blocks in what follows. Note that the fine tuning of the definition of the code blocks detailed in Table 2 is discussed in Section 9 on page 144.

5.8 noAdditionalIndent and indentRules

`latexindent.pl` operates on files by looking for code blocks, as detailed in Section 5.7; for each type of code block in Table 2 on the following page (which we will call a ⟨*thing*⟩ in what follows) it searches YAML fields for information in the following order:

1. `noAdditionalIndent` for the *name* of the current ⟨*thing*⟩;
2. `indentRules` for the *name* of the current ⟨*thing*⟩;

TABLE 2: Code blocks known to `latexindent.pl`

Code block	characters allowed in name	example

environments	`a-zA-Z@\*0-9_\\`	`\begin{myenv}`
		`body of myenv`
		`\end{myenv}`

optionalArguments	*inherits* name from parent (e.g environment name)	`[`
		`opt arg text`
		`]`

mandatoryArguments	*inherits* name from parent (e.g environment name)	`{`
		`mand arg text`
		`}`

commands	`+a-zA-Z@\*0-9_\:`	`\mycommand`⟨*arguments*⟩
keyEqualsValuesBracesBrackets	`a-zA-Z@\*0-9_\/.\h\{\}:\#-`	`my key/.style=`⟨*arguments*⟩
namedGroupingBracesBrackets	`0-9\.a-zA-Z@\*><`	`in`⟨*arguments*⟩
UnNamedGroupingBracesBrackets	*No name!*	`{` or `[` or `,` or `\&` or `)` or `(` or `$` followed by ⟨*arguments*⟩

ifElseFi	`@a-zA-Z` but must begin with either `\if` of `\@if`	`\ifnum...`
		`...`
		`\else`
		`...`
		`\fi`

items	User specified, see Listings 133 and 136 on page 46 and on page 47	`\begin{enumerate}`
		`  \item ...`
		`\end{enumerate}`

specialBeginEnd	User specified, see Listing 137 on page 47	`\[`
		`  ...`
		`\]`

afterHeading	User specified, see Listing 164 on page 53	`\chapter{title}`
		`  ...`
		`\section{title}`

filecontents	User specified, see Listing 53 on page 32	`\begin{filecontents}`
		`...`
		`\end{filecontents}`

3. `noAdditionalIndentGlobal` for the *type* of the current ⟨*thing*⟩;
4. `indentRulesGlobal` for the *type* of the current ⟨*thing*⟩.

Using the above list, the first piece of information to be found will be used; failing that, the value of `defaultIndent` is used. If information is found in multiple fields, the first one according to the list above will be used; for example, if information is present in both `indentRules` and in `noAdditionalIndentGlobal`, then the information from `indentRules` takes priority.

We now present details for the different type of code blocks known to `latexindent.pl`, as detailed in Table 2 on the previous page; for reference, there follows a list of the code blocks covered.

5.8.1 Environments and their arguments

There are a few different YAML switches governing the indentation of environments; let's start with the code shown in Listing 173.

LISTING 173: `myenv.tex`

```
\begin{outer}
\begin{myenv}
  body of environment
body of environment
     body of environment
\end{myenv}
\end{outer}
```

`noAdditionalIndent`: ⟨*fields*⟩

example 43 If we do not wish `myenv` to receive any additional indentation, we have a few choices available to us, as demonstrated in Listings 174 and 175.

LISTING 174: `myenv-noAdd1.yaml`

```
noAdditionalIndent:
    myenv: 1
```

LISTING 175: `myenv-noAdd2.yaml`

```
noAdditionalIndent:
    myenv:
        body: 1
```

On applying either of the following commands, ■

```
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd1.yaml
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd2.yaml
```

we obtain the output given in Listing 176; note in particular that the environment `myenv` has not received any *additional* indentation, but that the `outer` environment *has* still received indentation.

LISTING 176: `myenv.tex` output (using either Listing 174 or Listing 175)

```
\begin{outer}
   \begin{myenv}
   body of environment
   body of environment
   body of environment
   \end{myenv}
\end{outer}
```

example 44 Upon changing the YAML files to those shown in Listings 177 and 178, and running either

```
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd3.yaml
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd4.yaml
```

we obtain the output given in Listing 179.

LISTING 177: `myenv-noAdd3.yaml`

```
noAdditionalIndent:
    myenv: 0
```

LISTING 178: `myenv-noAdd4.yaml`

```
noAdditionalIndent:
    myenv:
        body: 0
```

LISTING 179: `myenv.tex output` (using either Listing 177 or Listing 178)

```
\begin{outer}
   \begin{myenv}
      body of environment
      body of environment
      body of environment
   \end{myenv}
\end{outer}
```

example 45 Let's now allow `myenv` to have some optional and mandatory arguments, as in Listing 180.

LISTING 180: `myenv-args.tex`

```
\begin{outer}
\begin{myenv}[%
  optional argument text
        optional argument text]%
  { mandatory argument text
 mandatory argument text}
  body of environment
body of environment
     body of environment
\end{myenv}
\end{outer}
```

Upon running

```
cmh:~$ latexindent.pl -l=myenv-noAdd1.yaml myenv-args.tex
```

we obtain the output shown in Listing 181; note that the optional argument, mandatory argument and body *all* have received no additional indent. This is because, when `noAdditionalIndent` is specified in ‘scalar’ form (as in Listing 174), then *all* parts of the environment (body, optional and mandatory arguments) are assumed to want no additional indent.

LISTING 181: `myenv-args.tex` using Listing 174

```
\begin{outer}
   \begin{myenv}[%
   optional argument text
   optional argument text]%
   { mandatory argument text
   mandatory argument text}
   body of environment
   body of environment
   body of environment
   \end{myenv}
\end{outer}
```

example 46 We may customise `noAdditionalIndent` for optional and mandatory arguments of the `myenv` environment, as shown in, for example, Listings 182 and 183.

LISTING 182: `myenv-noAdd5.yaml`

```
noAdditionalIndent:
    myenv:
        body: 0
        optionalArguments: 1
        mandatoryArguments: 0
```

LISTING 183: `myenv-noAdd6.yaml`

```
noAdditionalIndent:
    myenv:
        body: 0
        optionalArguments: 0
        mandatoryArguments: 1
```

Upon running

```
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd5.yaml
cmh:~$ latexindent.pl myenv.tex -l myenv-noAdd6.yaml
```

we obtain the respective outputs given in Listings 184 and 185. Note that in Listing 184 the text for the *optional* argument has not received any additional indentation, and that in Listing 185 the *mandatory* argument has not received any additional indentation; in both cases, the *body* has not received any additional indentation.

LISTING 184: `myenv-args.tex` using Listing 182

```
\begin{outer}
   \begin{myenv}[%
      optional argument text
      optional argument text]%
      { mandatory argument text
         mandatory argument text}
      body of environment
      body of environment
      body of environment
   \end{myenv}
\end{outer}
```

LISTING 185: `myenv-args.tex` using Listing 183

```
\begin{outer}
   \begin{myenv}[%
         optional argument text
         optional argument text]%
      { mandatory argument text
      mandatory argument text}
      body of environment
      body of environment
      body of environment
   \end{myenv}
\end{outer}
```

`indentRules`: ⟨*fields*⟩

example 47 We may also specify indentation rules for environment code blocks using the `indentRules` field; see, for example, Listings 186 and 187.

LISTING 186: `myenv-rules1.yaml`

```
indentRules:
    myenv: "    "
```

LISTING 187: `myenv-rules2.yaml`

```
indentRules:
    myenv:
        body: "    "
```

On applying either of the following commands,

```
cmh:~$ latexindent.pl myenv.tex -l myenv-rules1.yaml
cmh:~$ latexindent.pl myenv.tex -l myenv-rules2.yaml
```

we obtain the output given in Listing 188; note in particular that the environment `myenv` has received one tab (from the `outer` environment) plus three spaces from Listing 186 or 187.

LISTING 188: `myenv.tex` output (using either Listing 186 or Listing 187)

```
\begin{outer}
⎵\begin{myenv}
⎵␣␣␣body␣of␣environment
⎵␣␣␣body␣of␣environment
⎵␣␣␣body␣of␣environment
⎵\end{myenv}
\end{outer}
```

■

If you specify a field in `indentRules` using anything other than horizontal space, it will be ignored.

example 48 Returning to the example in Listing 180 that contains optional and mandatory arguments. Upon using Listing 186 as in

```
cmh:~$ latexindent.pl myenv-args.tex -l=myenv-rules1.yaml
```

we obtain the output in Listing 189; note that the body, optional argument and mandatory argument of `myenv` have *all* received the same customised indentation.

LISTING 189: `myenv-args.tex` using Listing 186

```
\begin{outer}
⎵\begin{myenv}[%
⎵␣␣␣␣␣␣optional␣argument␣text
⎵␣␣␣␣␣␣optional␣argument␣text]%
⎵␣␣␣{␣mandatory␣argument␣text
⎵␣␣␣␣␣␣mandatory␣argument␣text}
⎵␣␣␣body␣of␣environment
⎵␣␣␣body␣of␣environment
⎵␣␣␣body␣of␣environment
⎵\end{myenv}
\end{outer}
```

■

example 49 You can specify different indentation rules for the different features using, for example, Listings 190 and 191

■

Listing 190: myenv-rules3.yaml

```
indentRules:
    myenv:
        body: "   "
        optionalArguments: " "
```

Listing 191: myenv-rules4.yaml

```
indentRules:
    myenv:
        body: "   "
        mandatoryArguments:
    "\t\t"
```

After running

```
cmh:~$ latexindent.pl myenv-args.tex -l myenv-rules3.yaml
cmh:~$ latexindent.pl myenv-args.tex -l myenv-rules4.yaml
```

then we obtain the respective outputs given in Listings 192 and 193.

Listing 192: myenv-args.tex using Listing 190

```
\begin{outer}
⎵\begin{myenv}[%
⎵␣␣␣␣optional␣argument␣text
⎵␣␣␣␣optional␣argument␣text]%
⎵␣␣␣{␣mandatory␣argument␣text
⎵⎵␣␣␣mandatory␣argument␣text}
⎵␣␣␣body␣of␣environment
⎵␣␣␣body␣of␣environment
⎵␣␣␣body␣of␣environment
⎵\end{myenv}
\end{outer}
```

Listing 193: myenv-args.tex using Listing 191

```
\begin{outer}
⎵\begin{myenv}[%
⎵⎵␣␣␣optional␣argument␣text
⎵⎵␣␣␣optional␣argument␣text]%
⎵␣␣␣{␣mandatory␣argument␣text
⎵⎵⎵␣␣␣mandatory␣argument␣text}
⎵␣␣␣body␣of␣environment
⎵␣␣␣body␣of␣environment
⎵␣␣␣body␣of␣environment
⎵\end{myenv}
\end{outer}
```

Note that in Listing 192, the optional argument has only received a single space of indentation, while the mandatory argument has received the default (tab) indentation; the environment body has received three spaces of indentation.

In Listing 193, the optional argument has received the default (tab) indentation, the mandatory argument has received two tabs of indentation, and the body has received three spaces of indentation. ■

`noAdditionalIndentGlobal`: ⟨*fields*⟩

Assuming that your environment name is not found within neither `noAdditionalIndent` nor `indentRules`, the next place that `latexindent.pl` will look is `noAdditionalIndentGlobal`, and in particular *for the environments* key (see Listing 194).

Listing 194: noAdditionalIndentGlobal

```
noAdditionalIndentGlobal:
  environments: 0             # 0/1
```

example 50 Let's say that you change the value of `environments` to `1` in Listing 194, and that you run

```
cmh:~$ latexindent.pl myenv-args.tex -l env-noAdditionalGlobal.yaml
cmh:~$ latexindent.pl myenv-args.tex -l myenv-rules1.yaml,env-noAdditionalGlobal.yaml
```

The respective output from these two commands are in Listings 195 and 196; in Listing 195 notice that *both* environments receive no additional indentation but that the arguments of `myenv` still *do* receive indentation. In Listing 196 notice that the *outer* environment does not receive additional indentation, but because of the settings from `myenv-rules1.yaml` (in Listing 186 on the preceding page), the `myenv` environment still *does* receive indentation. ■

Listing 195: `myenv-args.tex` using Listing 194

```
\begin{outer}
\begin{myenv}[%
   optional argument text
   optional argument text]%
{ mandatory argument text
   mandatory argument text}
body of environment
body of environment
body of environment
\end{myenv}
\end{outer}
```

Listing 196: `myenv-args.tex` using Listings 186 and 194

```
\begin{outer}
\begin{myenv}[%
      optional argument text
      optional argument text]%
   { mandatory argument text
      mandatory argument text}
   body of environment
   body of environment
   body of environment
\end{myenv}
\end{outer}
```

example 51 In fact, `noAdditionalIndentGlobal` also contains keys that control the indentation of optional and mandatory arguments; on referencing Listings 197 and 198

Listing 197: `opt-args-no-add-glob.yaml`

```
noAdditionalIndentGlobal:
    optionalArguments: 1
```

Listing 198: `mand-args-no-add-glob.yaml`

```
noAdditionalIndentGlobal:
    mandatoryArguments: 1
```

we may run the commands

```
cmh:~$ latexindent.pl myenv-args.tex -local opt-args-no-add-glob.yaml
cmh:~$ latexindent.pl myenv-args.tex -local mand-args-no-add-glob.yaml
```

which produces the respective outputs given in Listings 199 and 200. Notice that in Listing 199 the *optional* argument has not received any additional indentation, and in Listing 200 the *mandatory* argument has not received any additional indentation.

Listing 199: `myenv-args.tex` using Listing 197

```
\begin{outer}
   \begin{myenv}[%
      optional argument text
      optional argument text]%
      { mandatory argument text
         mandatory argument text}
      body of environment
      body of environment
      body of environment
   \end{myenv}
\end{outer}
```

Listing 200: `myenv-args.tex` using Listing 198

```
\begin{outer}
   \begin{myenv}[%
         optional argument text
         optional argument text]%
      { mandatory argument text
      mandatory argument text}
      body of environment
      body of environment
      body of environment
   \end{myenv}
\end{outer}
```

`indentRulesGlobal`: ⟨*fields*⟩

The final check that `latexindent.pl` will make is to look for `indentRulesGlobal` as detailed in Listing 201.

Listing 201: `indentRulesGlobal`

```
indentRulesGlobal:
  environments: 0                   # 0/h-space
```

example 52 If you change the `environments` field to anything involving horizontal space, say `" "`, and then run the following commands

```
cmh:~$ latexindent.pl myenv-args.tex -l env-indentRules.yaml
cmh:~$ latexindent.pl myenv-args.tex -l myenv-rules1.yaml,env-indentRules.yaml
```

then the respective output is shown in Listings 202 and 203. Note that in Listing 202, both the environment blocks have received a single-space indentation, whereas in Listing 203 the `outer` environment has received single-space indentation (specified by `indentRulesGlobal`), but `myenv` has received `"   "`, as specified by the particular `indentRules` for `myenv` Listing 186 on page 60.

LISTING 202: `myenv-args.tex` using Listing 201

```
\begin{outer}
␣\begin{myenv}[%
␣␣␣␣␣optional␣argument␣text
␣␣␣␣␣optional␣argument␣text]%
␣␣{␣mandatory␣argument␣text
␣␣␣␣␣mandatory␣argument␣text}
␣␣body␣of␣environment
␣␣body␣of␣environment
␣␣body␣of␣environment
␣\end{myenv}
\end{outer}
```

LISTING 203: `myenv-args.tex` using Listings 186 and 201

```
\begin{outer}
␣\begin{myenv}[%
␣␣␣␣␣␣␣optional␣argument␣text
␣␣␣␣␣␣␣optional␣argument␣text]%
␣␣␣␣{␣mandatory␣argument␣text
␣␣␣␣␣␣␣mandatory␣argument␣text}
␣␣␣␣body␣of␣environment
␣␣␣␣body␣of␣environment
␣␣␣␣body␣of␣environment
␣\end{myenv}
\end{outer}
```

■

example 53 You can specify `indentRulesGlobal` for both optional and mandatory arguments, as detailed in Listings 204 and 205

LISTING 204: `opt-args-indent-rules-glob.yaml`

```
indentRulesGlobal:
    optionalArguments: "\t\t"
```

LISTING 205: `mand-args-indent-rules-glob.yaml`

```
indentRulesGlobal:
    mandatoryArguments: "\t\t"
```

Upon running the following commands

```
cmh:~$ latexindent.pl myenv-args.tex -local opt-args-indent-rules-glob.yaml
cmh:~$ latexindent.pl myenv-args.tex -local mand-args-indent-rules-glob.yaml
```

we obtain the respective outputs in Listings 206 and 207. Note that the *optional* argument in Listing 206 has received two tabs worth of indentation, while the *mandatory* argument has done so in Listing 207.

LISTING 206: `myenv-args.tex` using Listing 204

```
\begin{outer}
⎵\begin{myenv}[%
⎵⎵⎵⎵optional argument text
⎵⎵⎵⎵optional argument text]%
⎵⎵{ mandatory argument text
⎵⎵⎵mandatory argument text}
⎵⎵body of environment
⎵⎵body of environment
⎵⎵body of environment
⎵\end{myenv}
\end{outer}
```

LISTING 207: `myenv-args.tex` using Listing 205

```
\begin{outer}
⎵\begin{myenv}[%
⎵⎵⎵optional argument text
⎵⎵⎵optional argument text]%
⎵⎵{ mandatory argument text
⎵⎵⎵⎵mandatory argument text}
⎵⎵body of environment
⎵⎵body of environment
⎵⎵body of environment
⎵\end{myenv}
\end{outer}
```

■

5.8.2 Environments with items

With reference to Listings 133 and 136 on page 46 and on page 47, some commands may contain `item` commands; for the purposes of this discussion, we will use the code from Listing 134 on page 46.

Assuming that you've populated `itemNames` with the name of your `item`, you can put the item name into `noAdditionalIndent` as in Listing 208, although a more efficient approach may be to change the relevant field in `itemNames` to `0`.

example 54 Similarly, you can customise the indentation that your `item` receives using `indentRules`, as in Listing 209

LISTING 208: `item-noAdd1.yaml`

```
noAdditionalIndent:
    item: 1
# itemNames:
#   item: 0
```

LISTING 209: `item-rules1.yaml`

```
indentRules:
    item: " "
```

Upon running the following commands

```
cmh:~$ latexindent.pl items1.tex -local item-noAdd1.yaml
cmh:~$ latexindent.pl items1.tex -local item-rules1.yaml
```

the respective outputs are given in Listings 210 and 211; note that in Listing 210 that the text after each `item` has not received any additional indentation, and in Listing 211, the text after each `item` has received a single space of indentation, specified by Listing 209.

LISTING 210: `items1.tex` using Listing 208

```
\begin{itemize}
   \item some text here
   some more text here
   some more text here
   \item another item
   some more text here
\end{itemize}
```

LISTING 211: `items1.tex` using Listing 209

```
\begin{itemize}
␣␣␣\item␣some␣text␣here
␣␣␣␣some␣more␣text␣here
␣␣␣␣some␣more␣text␣here
␣␣␣\item␣another␣item
␣␣␣␣some␣more␣text␣here
\end{itemize}
```

■

example 55 Alternatively, you might like to populate `noAdditionalIndentGlobal` or `indentRulesGlobal` using the `items` key, as demonstrated in Listings 212 and 213. Note that there is a need to 'reset/remove' the `item` field from `indentRules` in both cases (see the hierarchy description given on page 55) as the `item` command is a member of `indentRules` by default.

LISTING 212: `items-noAdditionalGlobal.yaml`

```
indentRules:
    item: 0
noAdditionalIndentGlobal:
    items: 1
```

LISTING 213: `items-indentRulesGlobal.yaml`

```
indentRules:
    item: 0
indentRulesGlobal:
    items: " "
```

Upon running the following commands,

```
cmh:~$ latexindent.pl items1.tex -local items-noAdditionalGlobal.yaml
cmh:~$ latexindent.pl items1.tex -local items-indentRulesGlobal.yaml
```

the respective outputs from Listings 210 and 211 are obtained; note, however, that *all* such `item` commands without their own individual `noAdditionalIndent` or `indentRules` settings would behave as in these listings.

■

5.8.3 Commands with arguments

example 56 Let's begin with the simple example in Listing 214; when `latexindent.pl` operates on this file, the default output is shown in Listing 215. [a]

LISTING 214: `mycommand.tex`

```
\mycommand
{
mand arg text
mand arg text}
[
opt arg text
opt arg text
]
```

LISTING 215: `mycommand.tex` default output

```
\mycommand
{
   mand arg text
   mand arg text}
[
   opt arg text
   opt arg text
]
```

As in the environment-based case (see Listings 174 and 175 on page 57) we may specify `noAdditionalIndent` either in 'scalar' form, or in 'field' form, as shown in Listings 216 and 217

LISTING 216: `mycommand-noAdd1.yaml`

```
noAdditionalIndent:
    mycommand: 1
```

LISTING 217: `mycommand-noAdd2.yaml`

```
noAdditionalIndent:
    mycommand:
        body: 1
```

After running the following commands,

```
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd1.yaml
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd2.yaml
```

we receive the respective output given in Listings 218 and 219

LISTING 218: `mycommand.tex` using Listing 216

```
\mycommand
{
mand arg text
mand arg text}
[
opt arg text
opt arg text
]
```

LISTING 219: `mycommand.tex` using Listing 217

```
\mycommand
{
   mand arg text
   mand arg text}
[
   opt arg text
   opt arg text
]
```

Note that in Listing 218 that the 'body', optional argument *and* mandatory argument have *all* received no additional indentation, while in Listing 219, only the 'body' has not received any additional indentation. We define the 'body' of a command as any lines following the command name that include its optional or mandatory arguments.

---

[a] The command code blocks have quite a few subtleties, described in Section 5.9 on page 73.

■

example 57 We may further customise `noAdditionalIndent` for `mycommand` as we did in Listings 182 and 183 on page 59; explicit examples are given in Listings 220 and 221.

LISTING 220: `mycommand-noAdd3.yaml`

```
noAdditionalIndent:
    mycommand:
        body: 0
        optionalArguments: 1
        mandatoryArguments: 0
```

LISTING 221: `mycommand-noAdd4.yaml`

```
noAdditionalIndent:
    mycommand:
        body: 0
        optionalArguments: 0
        mandatoryArguments: 1
```

■

After running the following commands,

```
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd3.yaml
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd4.yaml
```

we receive the respective output given in Listings 222 and 223.

Listing 222: `mycommand.tex` using Listing 220

```
\mycommand
{
   mand arg text
   mand arg text}
[
opt arg text
opt arg text
]
```

Listing 223: `mycommand.tex` using Listing 221

```
\mycommand
{
mand arg text
mand arg text}
[
   opt arg text
   opt arg text
]
```

■

example 58 Attentive readers will note that the body of `mycommand` in both Listings 222 and 223 has received no additional indent, even though `body` is explicitly set to `0` in both Listings 220 and 221. This is because, by default, `noAdditionalIndentGlobal` for `commands` is set to `1` by default; this can be easily fixed as in Listings 224 and 225.

Listing 224: `mycommand-noAdd5.yaml`

```
noAdditionalIndent:
    mycommand:
        body: 0
        optionalArguments: 1
        mandatoryArguments: 0
noAdditionalIndentGlobal:
    commands: 0
```

Listing 225: `mycommand-noAdd6.yaml`

```
noAdditionalIndent:
    mycommand:
        body: 0
        optionalArguments: 0
        mandatoryArguments: 1
noAdditionalIndentGlobal:
    commands: 0
```

After running the following commands,

```
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd5.yaml
cmh:~$ latexindent.pl mycommand.tex -l mycommand-noAdd6.yaml
```

we receive the respective output given in Listings 226 and 227.

Listing 226: `mycommand.tex` using Listing 224

```
\mycommand
   {
      mand arg text
      mand arg text}
   [
   opt arg text
   opt arg text
   ]
```

Listing 227: `mycommand.tex` using Listing 225

```
\mycommand
   {
   mand arg text
   mand arg text}
   [
      opt arg text
      opt arg text
   ]
```

■

Both `indentRules` and `indentRulesGlobal` can be adjusted as they were for *environment* code blocks, as in Listings 190 and 191 on page 61 and Listings 201, 204 and 205 on pages 62–63.

5.8.4 ifelsefi code blocks

example 59 Let's use the simple example shown in Listing 228; when `latexindent.pl` operates on this file, the output as in Listing 229; note that the body of each of the `\if` statements have been indented, and that the `\else` statement has been accounted for correctly.

LISTING 228: `ifelsefi1.tex`

```
\ifodd\radius
\ifnum\radius<14
\pgfmathparse{100-(\radius)*4};
\else
\pgfmathparse{200-(\radius)*3};
\fi\fi
```

LISTING 229: `ifelsefi1.tex` default output

```
\ifodd\radius
	\ifnum\radius<14
		\pgfmathparse{100-(\radius)*4};
	\else
		\pgfmathparse{200-(\radius)*3};
	\fi\fi
```

It is recommended to specify `noAdditionalIndent` and `indentRules` in the 'scalar' form only for these type of code blocks, although the 'field' form would work, assuming that `body` was specified. Examples are shown in Listings 230 and 231.

LISTING 230: `ifnum-noAdd.yaml`

```
noAdditionalIndent:
    ifnum: 1
```

LISTING 231: `ifnum-indent-rules.yaml`

```
indentRules:
    ifnum: "	  "
```

After running the following commands,

```
cmh:~$ latexindent.pl ifelsefi1.tex -local ifnum-noAdd.yaml
cmh:~$ latexindent.pl ifelsefi1.tex -l ifnum-indent-rules.yaml
```

we receive the respective output given in Listings 232 and 233; note that in Listing 232, the `ifnum` code block has *not* received any additional indentation, while in Listing 233, the `ifnum` code block has received one tab and two spaces of indentation.

LISTING 232: `ifelsefi1.tex` using Listing 230

```
\ifodd\radius
	\ifnum\radius<14
	\pgfmathparse{100-(\radius)*4};
	\else
	\pgfmathparse{200-(\radius)*3};
	\fi\fi
```

LISTING 233: `ifelsefi1.tex` using Listing 231

```
\ifodd\radius
___\ifnum\radius<14
___␣␣\pgfmathparse{100-(\radius)*4};
___\else
___␣␣\pgfmathparse{200-(\radius)*3};
___\fi\fi
```

■

example 60 We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 234 and 235.

LISTING 234: `ifelsefi-noAdd-glob.yaml`

```
noAdditionalIndentGlobal:
    ifElseFi: 1
```

LISTING 235: `ifelsefi-indent-rules-global.yaml`

```
indentRulesGlobal:
    ifElseFi: " "
```

Upon running the following commands

```
cmh:~$ latexindent.pl ifelsefi1.tex -local ifelsefi-noAdd-glob.yaml
cmh:~$ latexindent.pl ifelsefi1.tex -l ifelsefi-indent-rules-global.yaml
```

we receive the outputs in Listings 236 and 237; notice that in Listing 236 neither of the `ifelsefi` code blocks have received indentation, while in Listing 237 both code blocks have received a single space of indentation.

■

LISTING 236: `ifelsefi1.tex` using Listing 234

```
\ifodd\radius
\ifnum\radius<14
\pgfmathparse{100-(\radius)*4};
\else
\pgfmathparse{200-(\radius)*3};
\fi\fi
```

LISTING 237: `ifelsefi1.tex` using Listing 235

```
\ifodd\radius
␣\ifnum\radius<14
␣␣\pgfmathparse{100-(\radius)*4};
␣\else
␣␣\pgfmathparse{200-(\radius)*3};
␣\fi\fi
```

■

example 61 U: 2018-04-27

We can further explore the treatment of `ifElseFi` code blocks in Listing 238, and the associated default output given in Listing 239; note, in particular, that the bodies of each of the 'or statements' have been indented.

LISTING 238: `ifelsefi2.tex`

```
\ifcase#1
zero%
\or
one%
\or
two%
\or
three%
\else
default
\fi
```

LISTING 239: `ifelsefi2.tex` default output

```
\ifcase#1
   zero%
\or
   one%
\or
   two%
\or
   three%
\else
   default
\fi
```

■

5.8.5 specialBeginEnd code blocks

Let's use the example from Listing 138 on page 48 which has default output shown in Listing 139 on page 48.

example 62 It is recommended to specify `noAdditionalIndent` and `indentRules` in the 'scalar' form for these type of code blocks, although the 'field' form would work, assuming that `body` was specified. Examples are shown in Listings 240 and 241.

LISTING 240: `displayMath-noAdd.yaml`

```
noAdditionalIndent:
    displayMath: 1
```

LISTING 241: `displayMath-indent-rules.yaml`

```
indentRules:
    displayMath: "\t\t\t"
```

After running the following commands,

```
cmh:~$ latexindent.pl special1.tex -local displayMath-noAdd.yaml
cmh:~$ latexindent.pl special1.tex -l displayMath-indent-rules.yaml
```

we receive the respective output given in Listings 242 and 243; note that in Listing 242, the `displayMath` code block has *not* received any additional indentation, while in Listing 243, the `displayMath` code block has received three tabs worth of indentation.

■

LISTING 242: `special1.tex` using Listing 240

```
The function $f$ has formula
\[
f(x)=x^2.
\]
If you like splitting dollars,
$
   g(x)=f(2x)
$
```

LISTING 243: `special1.tex` using Listing 241

```
The function $f$ has formula
\[
⎵⎵⎵f(x)=x^2.
\]
If you like splitting dollars,
$
⎵g(x)=f(2x)
$
```

■

example 63 We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 244 and 245.

LISTING 244: `special-noAdd-glob.yaml`

```
noAdditionalIndentGlobal:
    specialBeginEnd: 1
```

LISTING 245: `special-indent-rules-global.yaml`

```
indentRulesGlobal:
    specialBeginEnd: " "
```

Upon running the following commands

```
cmh:~$ latexindent.pl special1.tex -local special-noAdd-glob.yaml
cmh:~$ latexindent.pl special1.tex -l special-indent-rules-global.yaml
```

we receive the outputs in Listings 246 and 247; notice that in Listing 246 neither of the `special` code blocks have received indentation, while in Listing 247 both code blocks have received a single space of indentation.

LISTING 246: `special1.tex` using Listing 244

```
The function $f$ has formula
\[
f(x)=x^2.
\]
If you like splitting dollars,
$
g(x)=f(2x)
$
```

LISTING 247: `special1.tex` using Listing 245

```
The␣function␣$f$␣has␣formula
\[
␣f(x)=x^2.
\]
If␣you␣like␣splitting␣dollars,
$
␣g(x)=f(2x)
$
```

■

5.8.6 afterHeading code blocks

Let's use the example Listing 248 for demonstration throughout this Section. As discussed on page 54, by default `latexindent.pl` will not add indentation after headings.

LISTING 248: `headings2.tex`

```
\paragraph{paragraph
title}
paragraph text
paragraph text
```

example 64 On using the YAML file in Listing 250 by running the command

```
cmh:~$ latexindent.pl headings2.tex -l headings3.yaml
```

we obtain the output in Listing 249. Note that the argument of `paragraph` has received (default) indentation, and that the body after the heading statement has received (default) indentation. ■

LISTING 249: `headings2.tex` using Listing 250

```
\paragraph{paragraph
      title}
   paragraph text
   paragraph text
```

LISTING 250: `headings3.yaml`

```
indentAfterHeadings:
    paragraph:
       indentAfterThisHeading: 1
       level: 1
```

If we specify `noAdditionalIndent` as in Listing 252 and run the command

```
cmh:~$ latexindent.pl headings2.tex -l headings4.yaml
```

then we receive the output in Listing 251. Note that the arguments *and* the body after the heading of `paragraph` has received no additional indentation, because we have specified `noAdditionalIndent` in scalar form.

LISTING 251: `headings2.tex` using Listing 252

```
\paragraph{paragraph
title}
paragraph text
paragraph text
```

LISTING 252: `headings4.yaml`

```
indentAfterHeadings:
    paragraph:
       indentAfterThisHeading: 1
       level: 1
noAdditionalIndent:
    paragraph: 1
```

■

example 65 Similarly, if we specify `indentRules` as in Listing 254 and run analogous commands to those above, we receive the output in Listing 253; note that the *body*, *mandatory argument* and content *after the heading* of `paragraph` have *all* received three tabs worth of indentation.

LISTING 253: `headings2.tex` using Listing 254

```
\paragraph{paragraph
⎵⎵⎵⎵⎵⎵⎵⎵⎵title}
⎵⎵⎵paragraph text
⎵⎵⎵paragraph text
```

LISTING 254: `headings5.yaml`

```
indentAfterHeadings:
    paragraph:
       indentAfterThisHeading: 1
       level: 1
indentRules:
    paragraph: "\t\t\t"
```

■

example 66 We may, instead, specify `noAdditionalIndent` in 'field' form, as in Listing 256 which gives the output in Listing 255.

LISTING 255: `headings2.tex` using Listing 256

```
\paragraph{paragraph
   title}
paragraph text
paragraph text
```

LISTING 256: `headings6.yaml`

```
indentAfterHeadings:
    paragraph:
       indentAfterThisHeading: 1
       level: 1
noAdditionalIndent:
    paragraph:
        body: 0
        mandatoryArguments: 0
        afterHeading: 1
```

■

example 67 Analogously, we may specify `indentRules` as in Listing 258 which gives the output in Listing 257; note that mandatory argument text has only received a single space of indentation, while the body after the heading has received three tabs worth of indentation.

■

LISTING 257: `headings2.tex` using Listing 258

```
\paragraph{paragraph
⎵⎵⎵ title}
⎵⎵⎵paragraph text
⎵⎵⎵paragraph text
```

LISTING 258: `headings7.yaml`

```
indentAfterHeadings:
    paragraph:
       indentAfterThisHeading: 1
       level: 1
indentRules:
    paragraph:
        mandatoryArguments: " "
        afterHeading: "\t\t\t"
```

example 68 Finally, let's consider `noAdditionalIndentGlobal` and `indentRulesGlobal` shown in Listings 260 and 262 respectively, with respective output in Listings 259 and 261. Note that in Listing 260 the *mandatory argument* of `paragraph` has received a (default) tab's worth of indentation, while the body after the heading has received *no additional indentation*. Similarly, in Listing 261, the *argument* has received both a (default) tab plus two spaces of indentation (from the global rule specified in Listing 262), and the remaining body after `paragraph` has received just two spaces of indentation.

LISTING 259: `headings2.tex` using Listing 260

```
\paragraph{paragraph
   title}
paragraph text
paragraph text
```

LISTING 260: `headings8.yaml`

```
indentAfterHeadings:
    paragraph:
       indentAfterThisHeading: 1
       level: 1
noAdditionalIndentGlobal:
    afterHeading: 1
```

LISTING 261: `headings2.tex` using Listing 262

```
\paragraph{paragraph
⎵␣␣title}
␣␣paragraph␣text
␣␣paragraph␣text
```

LISTING 262: `headings9.yaml`

```
indentAfterHeadings:
    paragraph:
       indentAfterThisHeading: 1
       level: 1
indentRulesGlobal:
    afterHeading: "  "
```

5.8.7 The remaining code blocks

Referencing the different types of code blocks in Table 2 on page 56, we have a few code blocks yet to cover; these are very similar to the `commands` code block type covered comprehensively in Section 5.8.3 on page 65, but a small discussion defining these remaining code blocks is necessary.

5.8.7.1 keyEqualsValuesBracesBrackets

`latexindent.pl` defines this type of code block by the following criteria:

- it must immediately follow either `{` OR `[` OR `,` with comments and blank lines allowed.
- then it has a name made up of the characters detailed in Table 2 on page 56;
- then an $=$ symbol;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

N: 2019-07-13

See the `keyEqualsValuesBracesBrackets: follow` and `keyEqualsValuesBracesBrackets: name` fields of the fine tuning section in Listing 566 on page 144

example 69 An example is shown in Listing 263, with the default output given in Listing 264.

LISTING 263: `pgfkeys1.tex`

```
\pgfkeys{/tikz/.cd,
start coordinate/.initial={0,
\vertfactor},
}
```

LISTING 264: `pgfkeys1.tex` default output

```
\pgfkeys{/tikz/.cd,
␣start coordinate/.initial={0,
␣␣␣\vertfactor},
}
```

In Listing 264, note that the maximum indentation is three tabs, and these come from:

- the `\pgfkeys` command's mandatory argument;
- the `start coordinate/.initial` key's mandatory argument;
- the `start coordinate/.initial` key's body, which is defined as any lines following the name of the key that include its arguments. This is the part controlled by the *body* field for `noAdditionalIndent` and friends from page 55.

■

5.8.7.2 namedGroupingBracesBrackets

This type of code block is mostly motivated by tikz-based code; we define this code block as follows:

- it must immediately follow either *horizontal space* OR *one or more line breaks* OR `{` OR `[` OR `$` OR `)` OR `(`
- the name may contain the characters detailed in Table 2 on page 56;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

N: 2019-07-13

See the `NamedGroupingBracesBrackets: follow` and `NamedGroupingBracesBrackets: name` fields of the fine tuning section in Listing 566 on page 144

example 70 A simple example is given in Listing 265, with default output in Listing 266.

LISTING 265: `child1.tex`

```
\coordinate
child[grow=down]{
edge from parent [antiparticle]
node [above=3pt] {$C$}
}
```

LISTING 266: `child1.tex` default output

```
\coordinate
child[grow=down]{
␣␣edge from parent [antiparticle]
␣␣node [above=3pt] {$C$}
␣}
```

In particular, `latexindent.pl` considers `child`, `parent` and `node` all to be `namedGroupingBracesBrackets`[a]. Referencing Listing 266, note that the maximum indentation is two tabs, and these come from:

- the `child`'s mandatory argument;
- the `child`'s body, which is defined as any lines following the name of the `namedGroupingBracesBrackets` that include its arguments. This is the part controlled by the *body* field for `noAdditionalIndent` and friends from page 55.

---

[a] You may like to verify this by using the `-tt` option and checking `indent.log`!

■

5.8.7.3 UnNamedGroupingBracesBrackets

occur in a variety of situations; specifically, we define this type of code block as satisfying the following criteria:

- it must immediately follow either `{` OR `[` OR `,` OR `&` OR `)` OR `(` OR `$`;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

N: 2019-07-13

See the `UnNamedGroupingBracesBrackets: follow` field of the fine tuning section in Listing 566 on page 144

example 71 An example is shown in Listing 267 with default output give in Listing 268.

Listing 267: `psforeach1.tex`

```
\psforeach{\row}{%
{
{3,2.8,2.7,3,3.1}},%
{2.8,1,1.2,2,3},%
}
```

Listing 268: `psforeach1.tex` default output

```
\psforeach{\row}{%
␣{
␣␣␣{3,2.8,2.7,3,3.1}},%
␣{2.8,1,1.2,2,3},%
}
```

Referencing Listing 268, there are *three* sets of unnamed braces. Note also that the maximum value of indentation is three tabs, and these come from:

- the `\psforeach` command's mandatory argument;
- the *first* un-named braces mandatory argument;
- the *first* un-named braces *body*, which we define as any lines following the first opening `{` or `[` that defined the code block. This is the part controlled by the *body* field for `noAdditionalIndent` and friends from page 55.

■

Users wishing to customise the mandatory and/or optional arguments on a *per-name* basis for the `UnNamedGroupingBracesBrackets` should use `always-un-named`.

5.8.7.4 filecontents

code blocks behave just as `environments`, except that neither arguments nor items are sought.

5.8.8 Summary

Having considered all of the different types of code blocks, the functions of the fields given in Listings 269 and 270 should now make sense.

Listing 269: `noAdditionalIndentGlobal`

```
noAdditionalIndentGlobal:
  environments: 0                   # 0/1
  commands: 1                       # 0/1
  optionalArguments: 0              # 0/1
  mandatoryArguments: 0             # 0/1
  ifElseFi: 0                       # 0/1
  items: 0                          # 0/1
  keyEqualsValuesBracesBrackets: 0  # 0/1
  namedGroupingBracesBrackets: 0    # 0/1
  UnNamedGroupingBracesBrackets: 0  # 0/1
  specialBeginEnd: 0                # 0/1
  afterHeading: 0                   # 0/1
  filecontents: 0                   # 0/1
```

Listing 270: `indentRulesGlobal`

```
indentRulesGlobal:
  environments: 0                   #
    0/h-space
  commands: 0                       #
    0/h-space
  optionalArguments: 0              #
    0/h-space
  mandatoryArguments: 0             #
    0/h-space
  ifElseFi: 0                       #
    0/h-space
  items: 0                          #
    0/h-space
  keyEqualsValuesBracesBrackets: 0  #
    0/h-space
  namedGroupingBracesBrackets: 0    #
    0/h-space
  UnNamedGroupingBracesBrackets: 0  #
    0/h-space
  specialBeginEnd: 0                #
    0/h-space
  afterHeading: 0                   #
    0/h-space
  filecontents: 0                   #
    0/h-space
```

5.9 Commands and the strings between their arguments

The `command` code blocks will always look for optional (square bracketed) and mandatory (curly braced) arguments which can contain comments, line breaks and 'beamer' commands `<.*?>` between them. There are switches that can allow them to contain other strings, which we discuss next.

`commandCodeBlocks`: ⟨*fields*⟩

U: 2018-04-27

The `commandCodeBlocks` field contains a few switches detailed in Listing 271.

LISTING 271: `commandCodeBlocks`

```
commandCodeBlocks:
  roundParenthesesAllowed: 1
  stringsAllowedBetweenArguments:
    - amalgamate: 1
    - node
    - at
    - to
    - decoration
    - \+\+
    - \-\-
    - \#\#\d
  commandNameSpecial:
    - amalgamate: 1
    - ’@ifnextchar\[’

# change dos line breaks into unix
```

`roundParenthesesAllowed`: **0|1**

example 72 The need for this field was mostly motivated by commands found in code used to generate images in `PSTricks` and `tikz`; for example, let’s consider the code given in Listing 272.

LISTING 272: `pstricks1.tex`

```
\defFunction[algebraic]{torus}(u,v)
{(2+cos(u))*cos(v+\Pi)}
{(2+cos(u))*sin(v+\Pi)}
{sin(u)}
```

LISTING 273: `pstricks1` default output

```
\defFunction[algebraic]{torus}(u,v)
{(2+cos(u))*cos(v+\Pi)}
{(2+cos(u))*sin(v+\Pi)}
{sin(u)}
```

Notice that the `\defFunction` command has an optional argument, followed by a mandatory argument, followed by a round-parenthesis argument, (u, v).

By default, because `roundParenthesesAllowed` is set to 1 in Listing 271, then `latexindent.pl` will allow round parenthesis between optional and mandatory arguments. In the case of the code in Listing 272, `latexindent.pl` finds *all* the arguments of `defFunction`, both before and after `(u,v)`.

The default output from running `latexindent.pl` on Listing 272 actually leaves it unchanged (see Listing 273); note in particular, this is because of `noAdditionalIndentGlobal` as discussed on page 66.

Upon using the YAML settings in Listing 275, and running the command

```
cmh:~$ latexindent.pl pstricks1.tex -l noRoundParentheses.yaml
```

we obtain the output given in Listing 274.

■

LISTING 274: `pstricks1.tex` using Listing 275

```
\defFunction[algebraic]{torus}(u,v)
{(2+cos(u))*cos(v+\Pi)}
   {(2+cos(u))*sin(v+\Pi)}
   {sin(u)}
```

LISTING 275: `noRoundParentheses.yaml`

```
commandCodeBlocks:
    roundParenthesesAllowed: 0
```

Notice the difference between Listing 273 and Listing 274; in particular, in Listing 274, because round parentheses are *not* allowed, `latexindent.pl` finds that the `\defFunction` command finishes at the first opening round parenthesis. As such, the remaining braced, mandatory, arguments are found to be `UnNamedGroupingBracesBrackets` (see Table 2 on page 56) which, by default, assume indentation for their body, and hence the tabbed indentation in Listing 274. ■

example 73 Let's explore this using the YAML given in Listing 277 and run the command

```
cmh:~$ latexindent.pl pstricks1.tex -l defFunction.yaml
```

then the output is as in Listing 276.

LISTING 276: `pstricks1.tex` using Listing 277

```
\defFunction[algebraic]{torus}(u,v)
␣{(2+cos(u))*cos(v+\Pi)}
␣{(2+cos(u))*sin(v+\Pi)}
␣{sin(u)}
```

LISTING 277: `defFunction.yaml`

```
indentRules:
    defFunction:
        body: " "
```

Notice in Listing 276 that the *body* of the `defFunction` command i.e, the subsequent lines containing arguments after the command name, have received the single space of indentation specified by Listing 277. ■

`stringsAllowedBetweenArguments`: ⟨*fields*⟩

example 74 `tikz` users may well specify code such as that given in Listing 278; processing this code using `latexindent.pl` gives the default output in Listing 279.

LISTING 278: `tikz-node1.tex`

```
\draw[thin]
(c) to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]
```

LISTING 279: `tikz-node1` default output

```
\draw[thin]
(c) to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]
```

With reference to Listing 271 on the previous page, we see that the strings

> to, node, ++

are all allowed to appear between arguments; importantly, you are encouraged to add further names to this field as necessary. This means that when `latexindent.pl` processes Listing 278, it consumes:

- the optional argument `[thin]`
- the round-bracketed argument `(c)` because `roundParenthesesAllowed` is 1 by default
- the string `to` (specified in `stringsAllowedBetweenArguments`)
- the optional argument `[in=110,out=-90]`
- the string `++` (specified in `stringsAllowedBetweenArguments`) ■

- the round-bracketed argument `(0,-0.5cm)` because `roundParenthesesAllowed` is 1 by default
- the string `node` (specified in `stringsAllowedBetweenArguments`)
- the optional argument `[below,align=left,scale=0.5]` ■

example 75 We can explore this further, for example using Listing 281 and running the command

```
cmh:~$ latexindent.pl tikz-node1.tex -l draw.yaml
```

we receive the output given in Listing 280.

LISTING 280: `tikz-node1.tex` using Listing 281

```
\draw[thin]
␣␣(c)␣to[in=110,out=-90]
␣␣++(0,-0.5cm)
␣␣node[below,align=left,scale=0.5]
```

LISTING 281: `draw.yaml`

```
indentRules:
    draw:
        body: "  "
```

Notice that each line after the `\draw` command (its 'body') in Listing 280 has been given the appropriate two-spaces worth of indentation specified in Listing 281.

Let's compare this with the output from using the YAML settings in Listing 283, and running the command

```
cmh:~$ latexindent.pl tikz-node1.tex -l no-strings.yaml
```

given in Listing 282.

LISTING 282: `tikz-node1.tex` using Listing 283

```
\draw[thin]
(c) to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]
```

LISTING 283: `no-strings.yaml`

```
commandCodeBlocks:

    stringsAllowedBetweenArguments:
    0
```

In this case, `latexindent.pl` sees that:

- the `\draw` command finishes after the `(c)`, as `stringsAllowedBetweenArguments` has been set to 0 so there are no strings allowed between arguments;
- it finds a `namedGroupingBracesBrackets` called `to` (see Table 2 on page 56) *with* argument `[in=110,out=-90]`
- it finds another `namedGroupingBracesBrackets` but this time called `node` with argument `[below,align=left,scale=0.5]` ■

U: 2018-04-27

Referencing Listing 271 on page 74, , we see that the first field in the `stringsAllowedBetweenArguments` is `amalgamate` and is set to `1` by default. This is for users who wish to specify their settings in multiple YAML files. For example, by using the settings in either Listing 284 orListing 285 is equivalent to using the settings in Listing 286.

LISTING 284: `amalgamate-demo.yaml`

```
commandCodeBlocks:

    stringsAllowedBetweenArguments:
      - ’more’
      - ’strings’
      - ’here’
```

LISTING 285: `amalgamate-demo1.yaml`

```
commandCodeBlocks:

    stringsAllowedBetweenArguments:
      -
        amalgamate: 1
      - ’more’
      - ’strings’
      - ’here’
```

LISTING 286: `amalgamate-demo2.yaml`

```
commandCodeBlocks:

    stringsAllowedBetweenArguments:
      -
        amalgamate: 1
      - ’node’
      - ’at’
      - ’to’
      - ’decoration’
      - ’\+\+’
      - ’\-\-’
      - ’more’
      - ’strings’
      - ’here’
```

We specify `amalgamate` to be set to `0` and in which case any settings loaded prior to those specified, including the default, will be overwritten. For example, using the settings in Listing 287 means that only the strings specified in that field will be used.

LISTING 287: `amalgamate-demo3.yaml`

```
commandCodeBlocks:
    stringsAllowedBetweenArguments:
      -
        amalgamate: 0
      - ’further’
      - ’settings’
```

It is important to note that the `amalgamate` field, if used, must be in the first field, and specified using the syntax given in Listings 285 to 287.

example 76 We may explore this feature further with the code in Listing 288, whose default output is given in Listing 289.

LISTING 288: `for-each.tex`

```
\foreach \x/\y in {0/1,1/2}{
body of foreach
}
```

LISTING 289: `for-each` default output

```
\foreach \x/\y in {0/1,1/2}{
      body of foreach
   }
```

Let’s compare this with the output from using the YAML settings in Listing 291, and running the command

```
cmh:~$ latexindent.pl for-each.tex -l foreach.yaml
```

given in Listing 290.

LISTING 290: `for-each.tex` using Listing 291

```
\foreach \x/\y in {0/1,1/2}{
   body of foreach
}
```

LISTING 291: `foreach.yaml`

```
commandCodeBlocks:

    stringsAllowedBetweenArguments:
      -
        amalgamate: 0
      - ’\\x\/\\y’
      - ’in’
```

You might like to compare the output given in Listing 289 and Listing 290. Note,in particular, in

Listing 289 that the `foreach` command has not included any of the subsequent strings, and that the braces have been treated as a `namedGroupingBracesBrackets`. In Listing 290 the `foreach` command has been allowed to have `\x/\y` and `in` between arguments because of the settings given in Listing 291.

■

`commandNameSpecial`: ⟨*fields*⟩

U: 2018-04-27

There are some special command names that do not fit within the names recognised by `latexindent.pl`, the first one of which is `\@ifnextchar[`. From the perspective of `latexindent.pl`, the whole of the text `\@ifnextchar[` is a command, because it is immediately followed by sets of mandatory arguments. However, without the `commandNameSpecial` field, `latexindent.pl` would not be able to label it as such, because the `[` is, necessarily, not matched by a closing `]`.

example 77 For example, consider the sample file in Listing 292, which has default output in Listing 293.

LISTING 292: `ifnextchar.tex`

```
\parbox{
\@ifnextchar[{arg 1}{arg 2}
}
```

LISTING 293: `ifnextchar.tex` default output

```
\parbox{
   \@ifnextchar[{arg 1}{arg 2}
}
```

Notice that in Listing 293 the `parbox` command has been able to indent its body, because `latexindent.pl` has successfully found the command `\@ifnextchar` first; the pattern-matching of `latexindent.pl` starts from *the inner most <thing> and works outwards*, discussed in more detail on page 127.

For demonstration, we can compare this output with that given in Listing 294 in which the settings from Listing 295 have dictated that no special command names, including the `\@ifnextchar[` command, should not be searched for specially; as such, the `parbox` command has been *unable* to indent its body successfully, because the `\@ifnextchar[` command has not been found.

LISTING 294: `ifnextchar.tex` using Listing 295

```
\parbox{
\@ifnextchar[{arg 1}{arg 2}
}
```

LISTING 295: `no-ifnextchar.yaml`

```
commandCodeBlocks:
    commandNameSpecial: 0
```

■

The `amalgamate` field can be used for `commandNameSpecial`, just as for `stringsAllowedBetweenArguments`. The same condition holds as stated previously, which we state again here:

Warning!

It is important to note that the `amalgamate` field, if used, in either `commandNameSpecial` or `stringsAllowedBetweenArguments` must be in the first field, and specified using the syntax given in Listings 285 to 287.

SECTION 6

The -m (modifylinebreaks) switch

All features described in this section will only be relevant if the `-m` switch is used.

`modifylinebreaks`: ⟨*fields*⟩

As of Version 3.0, `latexindent.pl` has the `-m` switch, which permits `latexindent.pl` to modify line breaks, according to the specifications in the `modifyLineBreaks` field. *The settings in this field will only be considered if the* `-m` *switch has been used.* A snippet of the default settings of this field is shown in Listing 296.

LISTING 296: `modifyLineBreaks`

`-m`

```
modifyLineBreaks:
  preserveBlankLines: 1                 # 0/1
  condenseMultipleBlankLinesInto: 1     # 0/1
```

Having read the previous paragraph, it should sound reasonable that, if you call `latexindent.pl` using the `-m` switch, then you give it permission to modify line breaks in your file, but let's be clear:

> **Warning!**
>
> If you call `latexindent.pl` with the `-m` switch, then you are giving it permission to modify line breaks. By default, the only thing that will happen is that multiple blank lines will be condensed into one blank line; many other settings are possible, discussed next.

`preserveBlankLines`: **0|1**

This field is directly related to *poly-switches*, discussed in Section 6.3. By default, it is set to `1`, which means that blank lines will be *protected* from removal; however, regardless of this setting, multiple blank lines can be condensed if `condenseMultipleBlankLinesInto` is greater than `0`, discussed next.

`condenseMultipleBlankLinesInto`: ⟨*positive integer*⟩

Assuming that this switch takes an integer value greater than `0`, `latexindent.pl` will condense multiple blank lines into the number of blank lines illustrated by this switch.

example 78 As an example, Listing 297 shows a sample file with blank lines; upon running

```
cmh:~$ latexindent.pl myfile.tex -m -o=+-mod1
```

the output is shown in Listing 298; note that the multiple blank lines have been condensed into one blank line, and note also that we have used the `-m` switch! ■

LISTING 297: mlb1.tex

```
before blank line



after blank line



after blank line
```

LISTING 298: mlb1-mod1.tex

```
before blank line

after blank line

after blank line
```

N: 2022-03-13

6.1 Text Wrapping

The text wrapping routine has been over-hauled as of V3.16; I hope that the interface is simpler, and most importantly, the results are better.

The complete settings for this feature are given in Listing 299.

LISTING 299: textWrapOptions

-m

```
textWrapOptions:
  columns: 0
  multipleSpacesToSingle: 1
  removeBlockLineBreaks: 1
  when: before                        # before/after
  comments:
    wrap: 0                           # 0/1
    inheritLeadingSpace: 0            # 0/1
  blocksFollow:
    headings: 1                       # 0/1
    commentOnPreviousLine: 1          # 0/1
    par: 1                            # 0/1
    blankLine: 1                      # 0/1
    verbatim: 1                       # 0/1
    filecontents: 1                   # 0/1
    other: \\\]|\\item(?:\h|\[)       # regex
  blocksBeginWith:
    A-Z: 1                            # 0/1
    a-z: 1                            # 0/1
    0-9: 0                            # 0/1
    other: 0                          # regex
  blocksEndBefore:
    commentOnOwnLine: 1               # 0/1
    verbatim: 1                       # 0/1
    filecontents: 1                   # 0/1
    other: \\begin\{|\\\[|\\end\{     # regex
  huge: overflow                      # forbid mid-word line breaks
  separator: ""
```

6.1.1 Text wrap: overview

An overview of how the text wrapping feature works:

1. the default value of `columns` is 0, which means that text wrapping will *not* happen by default;
2. it happens *after* verbatim blocks have been found;
3. it happens *after* the oneSentencePerLine routine (see Section 6.2);
4. it can happen *before* or *after* all of the other code blocks are found and does *not* operate on a per-code-block basis; when using `before` this means that, including indentation, you may receive a column width wider than that which you specify in `columns`, and in which case you probably wish to explore `after` in Section 6.1.7;

N: 2023-01-01

5. code blocks to be text wrapped will:

   (a) *follow* the fields specified in `blocksFollow`
   (b) *begin* with the fields specified in `blocksBeginWith`
   (c) *end* before the fields specified in `blocksEndBefore`
6. setting `columns` to a value > 0 will text wrap blocks by first removing line breaks, and then wrapping according to the specified value of `columns`;
7. setting `columns` to -1 will *only* remove line breaks within the text wrap block;
8. by default, the text wrapping routine will remove line breaks within text blocks because `removeBlockLineBreak` is set to 1; switch it to 0 if you wish to change this;
9. about trailing comments within text wrap blocks:
   (a) trailing comments that do *not* have leading space instruct the text wrap routine to connect the lines *without* space (see Listing 337);
   (b) multiple trailing comments will be connected at the end of the text wrap block (see Listing 341);
   (c) the number of spaces between the end of the text wrap block and the (possibly combined) trailing comments is determined by the spaces (if any) at the end of the text wrap block (see Listing 343);

N: 2023-01-01

10. trailing comments can receive text wrapping ; examples are shown in Section 6.1.8 and Section 6.2.9.

We demonstrate this feature using a series of examples.

6.1.2 Text wrap: simple examples

example 79 Let's use the sample text given in Listing 300.

LISTING 300: `textwrap1.tex`

```
Here  is a line of text that will be wrapped by latexindent.pl.

Here is  a line of text that will be wrapped by latexindent.pl.
```

We will change the value of `columns` in Listing 302 and then run the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml textwrap1.tex
```

then we receive the output given in Listing 301.

LISTING 301: `textwrap1-mod1.tex`

```
Here is a line of
text that will be
wrapped by
latexindent.pl.

Here is a line of
text that will be
wrapped by
latexindent.pl.
```

LISTING 302: `textwrap1.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 20
```

example 80 If we set `columns` to -1 then `latexindent.pl` remove line breaks within the text wrap block, and will *not* perform text wrapping. We can use this to undo text wrapping.

Starting from the file in Listing 301 and using the settings in Listing 303

LISTING 303: `textwrap1A.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: -1
```

and running

```
cmh:~$ latexindent.pl -m -l textwrap1A.yaml textwrap1-mod1.tex
```

gives the output in Listing 304.

LISTING 304: `textwrap1-mod1A.tex`

```
Here is a line of text that will be wrapped by latexindent.pl.

Here is a line of text that will be wrapped by latexindent.pl.
```

example 81 By default, the text wrapping routine will convert multiple spaces into single spaces. You can change this behaviour by flicking the switch `multipleSpacesToSingle` which we have done in Listing 306

Using the settings in Listing 306 and running

```
cmh:~$ latexindent.pl -m -l textwrap1B.yaml textwrap1-mod1.tex
```

gives the output in Listing 305.

LISTING 305: `textwrap1-mod1B.tex`

```
Here␣␣is␣a␣line␣of
text␣that␣will␣be
wrapped␣by
latexindent.pl.

Here␣is␣␣a␣line␣of
text␣that␣will␣be
wrapped␣by
latexindent.pl.
```

LISTING 306: `textwrap1B.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 20
        multipleSpacesToSingle: 0
```

We note that in Listing 305 the multiple spaces have *not* been condensed into single spaces.

6.1.3 Text wrap: `blocksFollow` examples

We examine the `blocksFollow` field of Listing 299.

example 82 Let's use the sample text given in Listing 307.

LISTING 307: `tw-headings1.tex`

```
\section{my heading}\label{mylabel1}
text to
    be
 wrapped from the first section
\subsection{subheading}
text to
    be
 wrapped from the first section
```

We note that Listing 307 contains the heading commands `section` and `subsection`. Upon running the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-headings1.tex
```

then we receive the output given in Listing 308.

LISTING 308: `tw-headings1-mod1.tex`

```
\section{my heading}\label{mylabel1}
text to be wrapped
from the first
section
\subsection{subheading}
text to be wrapped
from the first
section
```

We reference Listing 299 on page 81 and also Listing 164 on page 53:

- in Listing 299 the `headings` field is set to `1`, which instructs `latexindent.pl` to read the fields from Listing 164 on page 53, *regardless of the value of indentAfterThisHeading or level*;
- the default is to assume that the heading command can, optionally, be followed by a `label` command.

If you find scenarios in which the default value of `headings` does not work, then you can explore the `other` field.

We can turn off `headings` as in Listing 310 and then run

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,bf-no-headings.yaml tw-headings1.tex
```

gives the output in Listing 309, in which text wrapping has been instructed *not to happen* following headings.

LISTING 309: `tw-headings1-mod2.tex`

```
\section{my heading}\label{mylabel1}
text to
be
wrapped from the first section
\subsection{subheading}
text to
be
wrapped from the first section
```

LISTING 310: `bf-no-headings.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        blocksFollow:
           headings: 0
```

example 83 Let's use the sample text given in Listing 311.

LISTING 311: `tw-comments1.tex`

```
% trailing comment
text to
    be
 wrapped following first comment
% another comment
text to
    be
 wrapped following second comment
```

We note that Listing 311 contains trailing comments. Upon running the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-comments1.tex
```

then we receive the output given in Listing 312.

LISTING 312: tw-comments1-mod1.tex

```
% trailing comment
text to be wrapped
following first
comment
% another comment
text to be wrapped
following second
comment
```

With reference to Listing 299 on page 81 the `commentOnPreviousLine` field is set to `1`, which instructs `latexindent.pl` to find text wrap blocks after a comment on its own line.

We can turn off `comments` as in Listing 314 and then run

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,bf-no-comments.yaml tw-comments1.tex
```

gives the output in Listing 313, in which text wrapping has been instructed *not to happen* following comments on their own line.

LISTING 313: tw-comments1-mod2.tex

```
% trailing comment
text to
be
wrapped following first comment
% another comment
text to
be
wrapped following second comment
```

LISTING 314: bf-no-comments.yaml

-m

```
modifyLineBreaks:
    textWrapOptions:
        blocksFollow:
           commentOnPreviousLine: 0
```

Referencing Listing 299 on page 81 the `blocksFollow` fields `par`, `blankline`, `verbatim` and `filecontents` fields operate in analogous ways to those demonstrated in the above.

The `other` field of the `blocksFollow` can either be `0` (turned off) or set as a regular expression. The default value is set to `\\\]|\\item(?:\h|\[)` which can be translated to *backslash followed by a square bracket* or *backslash item followed by horizontal space or a square bracket*, or in other words, *end of display math* or an item command.

example 84 Let's use the sample text given in Listing 315.

LISTING 315: tw-disp-math1.tex

```
text to
    be
 wrapped before display math
 \[ y = x\]
text to
    be
 wrapped after display math
```

We note that Listing 315 contains display math. Upon running the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-disp-math1.tex
```

then we receive the output given in Listing 316.

LISTING 316: `tw-disp-math1-mod1.tex`

```
text to be wrapped
before display math
\[ y = x\]
text to be wrapped
after display math
```

With reference to Listing 299 on page 81 the `other` field is set to `\\\]`, which instructs `latexindent.pl` to find text wrap blocks after the end of display math.

We can turn off this switch as in Listing 318 and then run

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,bf-no-disp-math.yaml tw-disp-math1.tex
```

gives the output in Listing 317, in which text wrapping has been instructed *not to happen* following display math.

LISTING 317: `tw-disp-math1-mod2.tex`

```
text to be wrapped
before display math
\[ y = x\]
text to
be
wrapped after display math
```

LISTING 318: `bf-no-disp-math.yaml` `-m`

```
modifyLineBreaks:
    textWrapOptions:
        blocksFollow:
           other: 0
```

Naturally, you should feel encouraged to customise this as you see fit. ■

The `blocksFollow` field *deliberately* does not default to allowing text wrapping to occur after `begin environment` statements. You are encouraged to customize the `other` field to accommodate the environments that you would like to text wrap individually, as in the next example.

example 85 Let's use the sample text given in Listing 319.

LISTING 319: `tw-bf-myenv1.tex`

```
text to
    be
 wrapped before myenv environment
 \begin{myenv}
text to
    be
 wrapped within myenv environment
 \end{myenv}
text to
    be
 wrapped after myenv environment
```

We note that Listing 319 contains `myenv` environment. Upon running the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-bf-myenv1.tex
```

then we receive the output given in Listing 320. ■

Listing 320: `tw-bf-myenv1-mod1.tex`

```
text to be wrapped
before myenv
environment
\begin{myenv}
   text to
   be
   wrapped within myenv environment
\end{myenv}
text to
be
wrapped after myenv environment
```

We note that we have *not* received much text wrapping. We can turn do better by employing Listing 322 and then run

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,tw-bf-myenv.yaml tw-bf-myenv1.tex
```

which gives the output in Listing 321, in which text wrapping has been implemented across the file.

Listing 321: `tw-bf-myenv1-mod2.tex`

```
text to be wrapped
before myenv
environment
\begin{myenv}
   text to be wrapped
   within myenv
   environment
\end{myenv}
text to be wrapped
after myenv
environment
```

Listing 322: `tw-bf-myenv.yaml`

-m

```
modifyLineBreaks:
    textWrapOptions:
        blocksFollow:
           other: |-
             (?x)
                \\\]
                |
                \\item(?:\h|\[)
                |
                \\begin\{myenv\} # <--- new bit
                |                # <--- new bit
                \\end\{myenv\}   # <--- new bit
```

6.1.4 Text wrap: `blocksBeginWith` examples

We examine the `blocksBeginWith` field of Listing 299 with a series of examples.

example 86 By default, text wrap blocks can begin with the characters `a-z` and `A-Z`.

If we start with the file given in Listing 323

Listing 323: `tw-0-9.tex`

```
123 text to
    be
 wrapped before display math
 \[ y = x\]
456 text to
    be
 wrapped after display math
```

and run the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-0-9.tex
```

then we receive the output given in Listing 324 in which text wrapping has *not* occurred.

Listing 324: `tw-0-9-mod1.tex`

```
123 text to
be
wrapped before display math
\[ y = x\]
456 text to
be
wrapped after display math
```

We can allow paragraphs to begin with `0-9` characters by using the settings in Listing 326 and running

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,bb-0-9-yaml tw-0-9.tex
```

gives the output in Listing 325, in which text wrapping *has* happened.

Listing 325: `tw-0-9-mod2.tex`

```
123 text to be
wrapped before
display math
\[ y = x\]
456 text to be
wrapped after
display math
```

Listing 326: `bb-0-9.yaml.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        blocksBeginWith:
           0-9: 1
```

example 87 Let's now use the file given in Listing 327

Listing 327: `tw-bb-announce1.tex`

```
% trailing comment
\announce{announce text}
    and text
    to be
 wrapped before
  goes here
```

and run the command

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml tw-bb-announce1.tex
```

then we receive the output given in Listing 328 in which text wrapping has *not* occurred.

Listing 328: `tw-bb-announce1-mod1.tex`

```
% trailing comment
\announce{announce text}
and text
to be
wrapped before
goes here
```

We can allow `\announce` to be at the beginning of paragraphs by using the settings in Listing 330 and running

```
cmh:~$ latexindent.pl -m -l textwrap1.yaml,tw-bb-announce.yaml tw-bb-announce1.tex
```

gives the output in Listing 329, in which text wrapping *has* happened.

LISTING 329: tw-bb-announce1-mod2.tex

```
% trailing comment
\announce{announce
   text} and text to
be wrapped before
goes here
```

LISTING 330: tw-bb-announce.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        blocksBeginWith:
           other: '\\announce'
```

■

6.1.5 Text wrap: `blocksEndBefore` examples

We examine the `blocksEndBefore` field of Listing 299 with a series of examples.

example 88 Let's use the sample text given in Listing 331.

LISTING 331: tw-be-equation.tex

```
before
equation
text
\begin{align}
  1 & 2 \\
  3 & 4
\end{align}
after
equation
text
```

We note that Listing 331 contains an environment. Upon running the command

```
cmh:~$ latexindent.pl -m -l textwrap1A.yaml tw-be-equation.tex
```

then we receive the output given in Listing 332.

LISTING 332: tw-be-equation-mod1.tex

```
before equation text
\begin{align}
   1 & 2 \\
   3 & 4
\end{align}
after
equation
text
```

With reference to Listing 299 on page 81 the `other` field is set to `\\begin\{|\\\[|\\end\{`, which instructs `latexindent.pl` to *stop* text wrap blocks before `begin` statements, display math, and `end` statements.

We can turn off this switch as in Listing 333 and then run

```
cmh:~$ latexindent.pl -m -l textwrap1A.yaml,tw-be-equation.yaml tw-be-equation.tex
```

gives the output in Listing 334, in which text wrapping has been instructed *not* to stop at these statements.

■

Listing 333: tw-be-equation.yaml

-m

```
modifyLineBreaks:
    textWrapOptions:
        blocksEndBefore:
           other: 0
```

Listing 334: tw-be-equation-mod2.tex

```
before equation text \begin{align} 1 & 2 \\ 3 & 4 \end{align} after equation text
```

Naturally, you should feel encouraged to customise this as you see fit. ■

6.1.6 Text wrap: trailing comments and spaces

We explore the behaviour of the text wrap routine in relation to trailing comments using the following examples.

example 89 The file in Listing 335 contains a trailing comment which *does* have a space infront of it.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc1.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output given in Listing 336.

Listing 335: tw-tc1.tex

```
foo␣%
bar
```

Listing 336: tw-tc1-mod1.tex

```
foo bar%
```

■

example 90 The file in Listing 337 contains a trailing comment which does *not* have a space infront of it.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc2.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output in Listing 338.

Listing 337: tw-tc2.tex

```
foo%
bar
```

Listing 338: tw-tc2-mod1.tex

```
foobar%
```

We note that, because there is *not* a space before the trailing comment, that the lines have been joined *without* a space. ■

example 91 The file in Listing 339 contains multiple trailing comments.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc3.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output in Listing 340.

Listing 339: tw-tc3.tex

```
foo %1
bar%2
three
```

Listing 340: tw-tc3-mod1.tex

```
foo barthree%1%2
```

■

example 92 The file in Listing 341 contains multiple trailing comments.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc4.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output in Listing 342.

LISTING 341: tw-tc4.tex

```
foo %1
bar%2
three%3
```

LISTING 342: tw-tc4-mod1.tex

```
foo barthree%1%2%3
```

■

example 93 The file in Listing 343 contains multiple trailing comments.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc5.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output in Listing 344.

LISTING 343: tw-tc5.tex

```
foo%1
bar%2
three␣%3
```

LISTING 344: tw-tc5-mod1.tex

```
foobarthree␣%1%2%3
```

The space at the end of the text block has been preserved. ■

example 94 The file in Listing 345 contains multiple trailing comments.

Running the command

```
cmh:~$ latexindent.pl -m tw-tc6.tex -l textwrap1A.yaml -o=+-mod1
```

gives the output in Listing 346.

LISTING 345: tw-tc6.tex

```
foo%1
bar
```

LISTING 346: tw-tc6-mod1.tex

```
foobar␣%1
```

The space at the end of the text block has been preserved. ■

6.1.7 Text wrap: when before/after

N: 2023-01-01

The text wrapping routine operates, by default, `before` the code blocks have been found, but this can be changed to `after`:

- `before` means it is likely that the columns of wrapped text may *exceed* the value specified in `columns`;
- `after` means it columns of wrapped text should *not* exceed the value specified in `columns`.

We demonstrate this in the following examples. See also Section 6.2.8.

example 95 Let's begin with the file in Listing 347. ■

LISTING 347: `textwrap8.tex`

```
This paragraph
has line breaks throughout its paragraph;
we would like to combine
the textwrapping
and paragraph removal routine.
\begin{myenv}
This paragraph
has line breaks throughout its paragraph;
we would like to combine
the textwrapping
and paragraph removal routine.
\end{myenv}
```

Using the settings given in Listing 349 and running the command

```
cmh:~$ latexindent.pl textwrap8.tex -o=+-mod1.tex -l=tw-before1.yaml -m
```

gives the output given in Listing 348.

LISTING 348: `textwrap8-mod1.tex`

```
This paragraph has line breaks
throughout its paragraph; we would
like to combine the textwrapping
and paragraph removal routine.
\begin{myenv}
  This paragraph has line breaks
  throughout its paragraph; we would
  like to combine the textwrapping
  and paragraph removal routine.
\end{myenv}
----|----|----|----|----|----|----|----|
   5   10   15   20   25   30   35   40
```

LISTING 349: `tw-before1.yaml` -m

```
defaultIndent: '  '

modifyLineBreaks:
    textWrapOptions:
        columns: 35
        when: before  # <!-------
        blocksFollow:
          other: \\begin\{myenv\}
```

We note that, in Listing 348, that the wrapped text has *exceeded* the specified value of `columns` (35) given in Listing 349. We can affect this by changing `when`; we explore this next.

example 96 We continue working with Listing 347.

Using the settings given in Listing 351 and running the command

```
cmh:~$ latexindent.pl textwrap8.tex -o=+-mod2.tex -l=tw-after1.yaml -m
```

gives the output given in Listing 350.

LISTING 350: `textwrap8-mod2.tex`

```
This paragraph has line breaks
throughout its paragraph; we would
like to combine the textwrapping
and paragraph removal routine.
\begin{myenv}
  This paragraph has line breaks
  throughout its paragraph; we
  would like to combine the
  textwrapping and paragraph
  removal routine.
\end{myenv}
----|----|----|----|----|----|----|----|
   5   10   15   20   25   30   35   40
```

LISTING 351: `tw-after1.yaml` -m

```
defaultIndent: '  '

modifyLineBreaks:
    textWrapOptions:
        columns: 35
        when: after   # <!-------
        blocksFollow:
          other: \\begin\{myenv\}
```

We note that, in Listing 350, that the wrapped text has *obeyed* the specified value of `columns` (35) given in Listing 351. ■

6.1.8 Text wrap: wrapping comments

N: 2023-01-01

You can instruct `latexindent.pl` to apply text wrapping to comments ; we demonstrate this with examples, see also Section 6.2.9.

example 97 We use the file in Listing 352 which contains a trailing comment block.

LISTING 352: `textwrap9.tex`

```
My first sentence
% first comment
%   second
%third comment
%     fourth
```

Using the settings given in Listing 354 and running the command

```
cmh:~$ latexindent.pl textwrap9.tex -o=+-mod1.tex -l=wrap-comments1.yaml -m
```

gives the output given in Listing 353.

LISTING 353: `textwrap9-mod1.tex`

```
My first sentence
% first comment second third
% comment fourth
----|----|----|----|----|----|----|----|
   5   10   15   20   25   30   35   40
```

LISTING 354: `wrap-comments1.yaml` -m

```
modifyLineBreaks:
  textWrapOptions:
    columns: 35
    comments:
      wrap: 1  #<!-------
```

We note that, in Listing 353, that the comments have been *combined and wrapped* because of the annotated line specified in Listing 354. ■

example 98 We use the file in Listing 355 which contains a trailing comment block.

LISTING 355: `textwrap10.tex`

```
My first sentence
%    first comment
%   second
%third comment
%     fourth
```

Using the settings given in Listing 357 and running the command ■

```
cmh:~$ latexindent.pl textwrap10.tex -o=+-mod1.tex -l=wrap-comments1.yaml -m
```

gives the output given in Listing 356.

LISTING 356: textwrap10-mod1.tex

```
My first sentence
% first comment second third
% comment fourth
----|----|----|----|----|----|----|----|
   5   10   15   20   25   30   35   40
```

LISTING 357: wrap-comments1.yaml -m

```
modifyLineBreaks:
  textWrapOptions:
    columns: 35
    comments:
      wrap: 1  #<!-------
```

We note that, in Listing 356, that the comments have been *combined and wrapped* because of the annotated line specified in Listing 357, and that the space from the leading comment has not been inherited; we will explore this further in the next example. ■

example 99 We continue to use the file in Listing 355.

Using the settings given in Listing 359 and running the command

```
cmh:~$ latexindent.pl textwrap10.tex -o=+-mod2.tex -l=wrap-comments2.yaml -m
```

gives the output given in Listing 358.

LISTING 358: textwrap10-mod2.tex

```
My first sentence
%    first comment second third
%    comment fourth
----|----|----|----|----|----|----|----|
   5   10   15   20   25   30   35   40
```

LISTING 359: wrap-comments2.yaml -m

```
modifyLineBreaks:
  textWrapOptions:
    columns: 35
    comments:
      wrap: 1                #<!-------
      inheritLeadingSpace: 1 #<!-------
```

We note that, in Listing 358, that the comments have been *combined and wrapped* and that the leading space has been inherited because of the annotated lines specified in Listing 359. ■

6.1.9 Text wrap: huge, tabstop and separator

U: 2021-07-23

The default value of `huge` is `overflow`, which means that words will *not* be broken by the text wrapping routine, implemented by the `Text::Wrap` [48]. There are options to change the `huge` option for the `Text::Wrap` module to either `wrap` or `die`. Before modifying the value of `huge`, please bear in mind the following warning:

Warning!

Changing the value of `huge` to anything other than `overflow` will slow down `latexindent.pl` significantly when the `-m` switch is active.

Furthermore, changing `huge` means that you may have some words *or commands*(!) split across lines in your .tex file, which may affect your output. I do not recommend changing this field.

example 100 For example, using the settings in Listings 361 and 363 and running the commands

```
cmh:~$    latexindent.pl -m textwrap4.tex -o=+-mod2A -l textwrap2A.yaml
cmh:~$    latexindent.pl -m textwrap4.tex -o=+-mod2B -l textwrap2B.yaml
```

gives the respective output in Listings 360 and 362. ■

LISTING 360: `textwrap4-mod2A.tex`

```
He
re
is
a
li
ne
of
te
xt
.
```

LISTING 361: `textwrap2A.yaml` -m

```yaml
modifyLineBreaks:
  textWrapOptions:
    columns: 3
    huge: wrap
```

LISTING 362: `textwrap4-mod2B.tex`

```
Here
is
a
line
of
text.
```

LISTING 363: `textwrap2B.yaml` -m

```yaml
modifyLineBreaks:
  textWrapOptions:
    columns: 3
```

N: 2020-11-06

You can also specify the `tabstop` field as an integer value, which is passed to the text wrap module; see [48] for details.

example 101 Starting with the code in Listing 364 with settings in Listing 365, and running the command

```
cmh:~$     latexindent.pl -m textwrap-ts.tex -o=+-mod1 -l tabstop.yaml
```

gives the code given in Listing 366.

LISTING 364: `textwrap-ts.tex`

```
x       y
```

LISTING 365: `tabstop.yaml` -m

```yaml
modifyLineBreaks:
    textWrapOptions:
        columns: 80
        tabstop: 9
        multipleSpacesToSingle: 0
```

LISTING 366: `textwrap-ts-mod1.tex`

```
x       y
```

You can specify `separator`, `break` and `unexpand` options in your settings in analogous ways to those demonstrated in Listings 363 and 365, and they will be passed to the `Text::Wrap` module. I have not found a useful reason to do this; see [48] for more details.

6.2 oneSentencePerLine: modifying line breaks for sentences

N: 2018-01-13

You can instruct `latexindent.pl` to format your file so that it puts one sentence per line. Thank you to [7] for helping to shape and test this feature. The behaviour of this part of the script is controlled by the switches detailed in Listing 367, all of which we discuss next.

Listing 367: oneSentencePerLine

-m

```
oneSentencePerLine:
  manipulateSentences: 0              # 0/1
  removeSentenceLineBreaks: 1         # 0/1
  multipleSpacesToSingle: 1           # 0/1
  textWrapSentences: 0                # 1 disables main textWrap
  sentenceIndent: ""
  sentencesFollow:
    par: 1                            # 0/1
    blankLine: 1                      # 0/1
    fullStop: 1                       # 0/1
    exclamationMark: 1                # 0/1
    questionMark: 1                   # 0/1
    rightBrace: 1                     # 0/1
    commentOnPreviousLine: 1          # 0/1
    other: 0                          # regex
  sentencesBeginWith:
    A-Z: 1                            # 0/1
    a-z: 0                            # 0/1
    other: 0                          # regex
  sentencesEndWith:
    basicFullStop: 0                  # 0/1
    betterFullStop: 1                 # 0/1
    exclamationMark: 1                # 0/1
    questionMark: 1                   # 0/1
    other: 0                          # regex
  sentencesDoNOTcontain:
    other: \\begin|\\end              # regex
```

6.2.1 oneSentencePerLine: overview

An overview of how the oneSentencePerLine routine feature works:

1. the default value of `manipulateSentences` is 0, which means that oneSentencePerLine will *not* happen by default;
2. it happens *after* verbatim blocks have been found;
3. it happens *before* the text wrapping routine (see Section 6.1);
4. it happens *before* the main code blocks have been found;
5. sentences to be found:
    (a) *follow* the fields specified in `sentencesFollow`
    (b) *begin* with the fields specified in `sentencesBeginWith`
    (c) *end* with the fields specified in `sentencesEndWith`
6. by default, the oneSentencePerLine routine will remove line breaks within sentences because `removeBlockLineBreaks` is set to 1; switch it to 0 if you wish to change this;
7. sentences can be text wrapped according to `textWrapSentences`, and will be done either `before` or `after` the main indentation routine (see Section 6.2.8);
8. about trailing comments within text wrap blocks:
    (a) multiple trailing comments will be connected at the end of the sentence;
    (b) the number of spaces between the end of the sentence and the (possibly combined) trailing comments is determined by the spaces (if any) at the end of the sentence.

We demonstrate this feature using a series of examples.

`manipulateSentences`: **0|1**

This is a binary switch that details if `latexindent.pl` should perform the sentence manipulation routine; it is *off* (set to `0`) by default, and you will need to turn it on (by setting it to `1`) if you want the script to modify line breaks surrounding and within sentences.

`removeSentenceLineBreaks`: **0|1**

When operating upon sentences `latexindent.pl` will, by default, remove internal line breaks as `removeSentenceLineBreaks` is set to `1`. Setting this switch to `0` instructs `latexindent.pl` not to do so.

example 102 For example, consider `multiple-sentences.tex` shown in Listing 368.

LISTING 368: `multiple-sentences.tex`

```
This is the first
sentence. This is the; second, sentence. This is the
third sentence.

This is the fourth
sentence! This is the fifth sentence? This is the
sixth sentence.
```

If we use the YAML files in Listings 370 and 372, and run the commands

```
cmh:~$ latexindent.pl multiple-sentences -m -l=manipulate-sentences.yaml
cmh:~$ latexindent.pl multiple-sentences -m -l=keep-sen-line-breaks.yaml
```

then we obtain the respective output given in Listings 369 and 371.

LISTING 369: `multiple-sentences.tex` using Listing 370

```
This is the first sentence.
This is the; second, sentence.
This is the third sentence.

This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 370: `manipulate-sentences.yaml` -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
```

LISTING 371: `multiple-sentences.tex` using Listing 372

```
This is the first
sentence.
This is the; second, sentence.
This is the
third sentence.

This is the fourth
sentence!
This is the fifth sentence?
This is the
sixth sentence.
```

LISTING 372: `keep-sen-line-breaks.yaml` -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        removeSentenceLineBreaks: 0
```

Notice, in particular, that the 'internal' sentence line breaks in Listing 368 have been removed in Listing 369, but have not been removed in Listing 371. ■

`multipleSpacesToSingle`: **0|1**

U: 2022-03-25

By default, the one-sentence-per-line routine will convert multiple spaces into single spaces. You can change this behaviour by changing the switch `multipleSpacesToSingle` to a value of `0`.

The remainder of the settings displayed in Listing 367 on page 96 instruct `latexindent.pl` on how to define a sentence. From the perspective of `latexindent.pl` a sentence must:

- *follow* a certain character or set of characters (see Listing 373); by default, this is either `\par`, a blank line, a full stop/period (.), exclamation mark (!), question mark (?) right brace (}) or a comment on the previous line;
- *begin* with a character type (see Listing 374); by default, this is only capital letters;
- *end* with a character (see Listing 375); by default, these are full stop/period (.), exclamation mark (!) and question mark (?).

In each case, you can specify the `other` field to include any pattern that you would like; you can specify anything in this field using the language of regular expressions.

Listing 373: `sentencesFollow` -m

```
sentencesFollow:
  par: 1                                # 0/1
  blankLine: 1                          # 0/1
  fullStop: 1                           # 0/1
  exclamationMark: 1                    # 0/1
  questionMark: 1                       # 0/1
  rightBrace: 1                         # 0/1
  commentOnPreviousLine: 1              # 0/1
  other: 0                              # regex
```

Listing 374: `sentencesBeginWith` -m

```
sentencesBeginWith:
  A-Z: 1                                # 0/1
  a-z: 0                                # 0/1
  other: 0                              # regex
```

Listing 375: `sentencesEndWith` -m

```
sentencesEndWith:
  basicFullStop: 0                      # 0/1
  betterFullStop: 1                     # 0/1
  exclamationMark: 1                    # 0/1
  questionMark: 1                       # 0/1
  other: 0                              # regex
```

6.2.2 oneSentencePerLine: sentencesFollow

Let's explore a few of the switches in `sentencesFollow`.

example 103 We start with Listing 368 on the preceding page, and use the YAML settings given in Listing 377. Using the command

```
cmh:~$ latexindent.pl multiple-sentences -m -l=sentences-follow1.yaml
```

we obtain the output given in Listing 376.

Listing 376: `multiple-sentences.tex` using Listing 377

```
This is the first sentence.
This is the; second, sentence.
This is the third sentence.

This is the fourth
sentence!
This is the fifth sentence?
This is the sixth sentence.
```

Listing 377: `sentences-follow1.yaml` -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesFollow:
            blankLine: 0
```

■

Notice that, because `blankLine` is set to `0`, `latexindent.pl` will not seek sentences following a blank line, and so the fourth sentence has not been accounted for.

example 104 We can explore the `other` field in Listing 373 with the `.tex` file detailed in Listing 378.

LISTING 378: `multiple-sentences1.tex`

```
(Some sentences stand alone in brackets.) This is the first
sentence. This is the; second, sentence. This is the
third sentence.
```

Upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences1 -m -l=manipulate-sentences.yaml
cmh:~$ latexindent.pl multiple-sentences1 -m -l=manipulate-sentences.yaml,sentences-follow2.yaml
```

then we obtain the respective output given in Listings 379 and 380.

LISTING 379: `multiple-sentences1.tex` using Listing 370 on page 97

```
(Some sentences stand alone in brackets.) This is the first
sentence.
This is the; second, sentence.
This is the third sentence.
```

LISTING 380: `multiple-sentences1.tex` using Listing 381

```
(Some sentences stand alone in brackets.)
This is the first sentence.
This is the; second, sentence.
This is the third sentence.
```

LISTING 381: `sentences-follow2.yaml` `-m`

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesFollow:
           other: "\)"
```

Notice that in Listing 379 the first sentence after the `)` has not been accounted for, but that following the inclusion of Listing 381, the output given in Listing 380 demonstrates that the sentence *has* been accounted for correctly.

6.2.3 oneSentencePerLine: sentencesBeginWith

By default, `latexindent.pl` will only assume that sentences begin with the upper case letters `A-Z`; you can instruct the script to define sentences to begin with lower case letters (see Listing 374), and we can use the `other` field to define sentences to begin with other characters.

example 105 We use the file in Listing 382.

LISTING 382: `multiple-sentences2.tex`

```
This is the first
sentence.

$a$ can
represent a
number. 7 is
at the beginning of this sentence.
```

Upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences2 -m -l=manipulate-sentences.yaml
cmh:~$ latexindent.pl multiple-sentences2 -m -l=manipulate-sentences.yaml,sentences-begin1.yaml
```

then we obtain the respective output given in Listings 383 and 384.

LISTING 383: `multiple-sentences2.tex` using Listing 370 on page 97

```
This is the first sentence.

$a$ can
represent a
number. 7 is
at the beginning of this sentence.
```

LISTING 384: `multiple-sentences2.tex` using Listing 385

```
This is the first sentence.

$a$ can represent a number.
7 is at the beginning of this sentence.
```

LISTING 385: `sentences-begin1.yaml` -m

```yaml
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesBeginWith:
            other: "\$|[0-9]"
```

Notice that in Listing 383, the first sentence has been accounted for but that the subsequent sentences have not. In Listing 384, all of the sentences have been accounted for, because the `other` field in Listing 385 has defined sentences to begin with either `$` or any numeric digit, `0` to `9`.

■

6.2.4 oneSentencePerLine: sentencesEndWith

example 106 Let's return to Listing 368 on page 97; we have already seen the default way in which `latexindent.pl` will operate on the sentences in this file in Listing 369 on page 97. We can populate the `other` field with any character that we wish; for example, using the YAML specified in Listing 387 and the command

```
cmh:~$ latexindent.pl multiple-sentences -m -l=sentences-end1.yaml
cmh:~$ latexindent.pl multiple-sentences -m -l=sentences-end2.yaml
```

then we obtain the output in Listing 386.

LISTING 386: `multiple-sentences.tex` using Listing 387

```
This is the first sentence.
This is the;
second, sentence.
This is the third sentence.

This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 387: `sentences-end1.yaml` -m

```yaml
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesEndWith:
          other: "\:|\;|\,"
```

LISTING 388: `multiple-sentences.tex` using Listing 389

```
This is the first sentence.
This is the;
second,
sentence.
This is the third sentence.

This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 389: `sentences-end2.yaml` -m

```yaml
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesEndWith:
          other: "\:|\;|\,"
        sentencesBeginWith:
          a-z: 1
```

■

There is a subtle difference between the output in Listings 386 and 388; in particular, in Listing 386 the word `sentence` has not been defined as a sentence, because we have not instructed `latexindent.pl` to begin sentences with lower case letters. We have changed this by using the settings in Listing 389, and the associated output in Listing 388 reflects this.

Referencing Listing 375 on page 98, you'll notice that there is a field called `basicFullStop`, which is set to `0`, and that the `betterFullStop` is set to `1` by default.

example 107 Let's consider the file shown in Listing 390.

LISTING 390: `url.tex`

```
This sentence, \url{tex.stackexchange.com/} finishes here. Second sentence.
```

Upon running the following commands

```
cmh:~$ latexindent.pl url -m -l=manipulate-sentences.yaml
```

we obtain the output given in Listing 391.

LISTING 391: `url.tex` using Listing 370 on page 97

```
This sentence, \url{tex.stackexchange.com/} finishes here.
Second sentence.
```

Notice that the full stop within the url has been interpreted correctly. This is because, within the `betterFullStop`, full stops at the end of sentences have the following properties:

- they are ignored within `e.g.` and `i.e.`;
- they can not be immediately followed by a lower case or upper case letter;
- they can not be immediately followed by a hyphen, comma, or number.

N: 2019-07-13

If you find that the `betterFullStop` does not work for your purposes, then you can switch it off by setting it to `0`, and you can experiment with the `other` field. You can also seek to customise the `betterFullStop` routine by using the *fine tuning*, detailed in Listing 566 on page 144.

The `basicFullStop` routine should probably be avoided in most situations, as it does not accommodate the specifications above.

example 108 For example, using the following command

```
cmh:~$ latexindent.pl url -m -l=alt-full-stop1.yaml
```

and the YAML in Listing 393 gives the output in Listing 392.

LISTING 392: `url.tex` using Listing 393

```
This sentence, \url{tex.
   stackexchange.com/} finishes here.Second sentence.
```

LISTING 393: `alt-full-stop1.yaml` -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesEndWith:
            basicFullStop: 1
            betterFullStop: 0
```

Notice that the full stop within the URL has not been accommodated correctly because of the non-default settings in Listing 393.

6.2.5 oneSentencePerLine: sentencesDoNOTcontain

N: 2023-09-09

You can specify patterns that sentences do *not* contain using the field in Listing 394.

Listing 394: `sentencesDoNOTcontain` -m

```
  sentencesDoNOTcontain:
    other: \\begin|\\end              # regex
```

U: 2023-09-09

If sentences run across environments then, by default, they will *not* be considered a sentence by `latexindent.pl`.

example 109 For example, if we use the `.tex` file in Listing 395

Listing 395: `multiple-sentences4.tex`

```
This sentence
\begin{itemize}
  \item continues
\end{itemize}
across itemize
and finishes here.
```

and run the command

```
cmh:~$ latexindent.pl multiple-sentences4 -m -l=manipulate-sentences.yaml
```

then the output is unchanged, because the default value of `sentencesDoNOTcontain` says, *sentences do NOT contain*

This means that, by default, `latexindent.pl` does *not* consider the file in Listing 395 to have a sentence. `\\begin`

■

example 110 We can customise the `sentencesDoNOTcontain` field with anything that we do *not* want sentences to contain.

We begin with the file in Listing 396.

Listing 396: `sentence-dnc1.tex`

```
This should not be a sentence \cmh{?} and should not change.
But this
one should.
```

Upon running the following commands

```
cmh:~$ latexindent.pl sentence-dnc1.tex -m -l=dnc1.yaml
```

then we obtain the output given in Listing 397.

Listing 397: `sentence-dnc1-mod1.tex`

```
This should not be a sentence \cmh{?} and should not change.
But this one should.
```

Listing 398: `dnc1.yaml` -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesDoNOTcontain:
            other: |-
              (?x)
              \\begin
              |
              \\cmh
```

The settings in Listing 398 say that sentences do *not* contain `\begin` and that they do not contain `\cmh`

■

example 111 We can implement case insensitivity for the `sentencesDoNOTcontain` field.

We begin with the file in Listing 399.

LISTING 399: `sentence-dnc2.tex`

```
This should not be a sentence \cmh{?} and should not change.
This should not be a sentence \CMH{?} and should not change.
But this
one should.
```

Upon running the following commands

```
cmh:~$ latexindent.pl sentence-dnc2.tex -m -l=dnc2.yaml
```

then we obtain the output given in Listing 400.

LISTING 400: `sentence-dnc2-mod2.tex`

```
This should not be a sentence \cmh{?} and should not change.
This should not be a sentence \CMH{?} and should not change.
But this one should.
```

LISTING 401: `dnc2.yaml` -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesDoNOTcontain:
            other: |-
              (?xi)    #<!----
              \\begin
              |
              \\cmh
```

The settings in Listing 401 say that sentences do *not* contain `\begin` and that they do not contain *case insensitive* versions of `\cmh` ■

example 112 We can turn off `sentenceDoNOTcontain` by setting it to `0` as in Listing 402.

LISTING 402: `dnc-off.yaml` -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesDoNOTcontain: 0
```

The settings in Listing 402 mean that sentences can contain any character. ■

6.2.6 Features of the oneSentencePerLine routine

The sentence manipulation routine takes place *after* verbatim environments, preamble and trailing comments have been accounted for; this means that any characters within these types of code blocks will not be part of the sentence manipulation routine.

example 113 For example, if we begin with the `.tex` file in Listing 403, and run the command

```
cmh:~$ latexindent.pl multiple-sentences3 -m -l=manipulate-sentences.yaml
```

then we obtain the output in Listing 404. ■

LISTING 403: `multiple-sentences3.tex`

```
The first sentence continues after the verbatim
\begin{verbatim}
  there are sentences within this. These
  will not be operated
  upon by latexindent.pl.
\end{verbatim}
and finishes here. Second sentence % a commented full stop.
contains trailing comments,
which are ignored.
```

LISTING 404: `multiple-sentences3.tex` using Listing 370 on page 97

```
The first sentence continues after the verbatim \begin{verbatim}
  there are sentences within this. These
  will not be operated
  upon by latexindent.pl.
\end{verbatim} and finishes here.
Second sentence contains trailing comments, which are ignored.
% a commented full stop.
```

■

6.2.7 oneSentencePerLine: text wrapping and indenting sentences

N: 2018-08-13

The `oneSentencePerLine` can be instructed to perform text wrapping and indentation upon sentences.

example 114 Let's use the code in Listing 405.

LISTING 405: `multiple-sentences5.tex`

```
A distincao entre conteudo \emph{real} e conteudo \emph{intencional} esta
relacionada, ainda, a distincao entre o conceito husserliano de
\emph{experiencia} e o uso popular desse termo. No sentido comum,
o \term{experimentado} e um complexo de eventos exteriores,
e o \term{experimentar} consiste em percepcoes (alem de julgamentos e outros
atos) nas quais tais eventos aparecem como objetos, e objetos frequentemente
to the end.
```

Referencing Listing 407, and running the following command

```
cmh:~$ latexindent.pl multiple-sentences5 -m -l=sentence-wrap1.yaml
```

we receive the output given in Listing 406.

LISTING 406: `multiple-sentences5.tex` using Listing 407

```
A distincao entre conteudo \emph{real} e conteudo
  \emph{intencional} esta relacionada, ainda, a
  distincao entre o conceito husserliano de
  \emph{experiencia} e o uso popular desse termo.
No sentido comum, o \term{experimentado} e um
  complexo de eventos exteriores, e o
  \term{experimentar} consiste em percepcoes (alem
  de julgamentos e outros atos) nas quais tais
  eventos aparecem como objetos, e objetos
  frequentemente to the end.
```

LISTING 407: `sentence-wrap1.yaml` -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        removeSentenceLineBreaks: 1
        textWrapSentences: 1
        sentenceIndent: "  "
    textWrapOptions:
        columns: 50
```

■

If you specify `textWrapSentences` as 1, but do *not* specify a value for `columns` then the text wrapping will *not* operate on sentences, and you will see a warning in `indent.log`.

example 115 The indentation of sentences requires that sentences are stored as code blocks. This means that you may need to tweak Listing 375 on page 98. Let's explore this in relation to Listing 408.

LISTING 408: `multiple-sentences6.tex`

```
Consider the following:
\begin{itemize}
        \item firstly.
        \item secondly.
\end{itemize}
```

By default, `latexindent.pl` will find the full-stop within the first `item`, which means that, upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml
cmh:~$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml
    -y="modifyLineBreaks:oneSentencePerLine:sentenceIndent:’’"
```

we receive the respective output in Listing 409 and Listing 410.

LISTING 409: `multiple-sentences6-mod1.tex` using Listing 407

```
Consider the following:
\begin{itemize}
   \item firstly.
   \item secondly.
\end{itemize}
```

LISTING 410: `multiple-sentences6-mod2.tex` using Listing 407 and no sentence indentation

```
Consider the following:
\begin{itemize}
   \item firstly.
   \item secondly.
\end{itemize}
```

We note that Listing 409 the `itemize` code block has *not* been indented appropriately. This is because the oneSentencePerLine has been instructed to store sentences (because Listing 407); each sentence is then searched for code blocks. ■

example 116 We can tweak the settings in Listing 375 on page 98 to ensure that full stops are not followed by `item` commands, and that the end of sentences contains `\end{itemize}` as in Listing 411. This setting is actually an appended version of the `betterFullStop` from the `fineTuning`, detailed in Listing 566 on page 144. ■

LISTING 411: `itemize.yaml`

-m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 45
    oneSentencePerLine:
        sentencesEndWith:
            betterFullStop: 0
            other: |-
                (?x)
                (?:                             # new
                 (?:\R|\h)*\\item               # new
                )                               # new
                |
                (?:
                  \.\)
                  (?!\h*[a-z])
                )
                |
                (?:
                  (?<!
                    (?:
                      (?:[eE]\.[gG])
                      |
                      (?:[iI]\.[eE])
                      |
                      (?:etc)
                    )
                  )
                )
                \.
                (?:\h*\R*(?:\\end\{itemize\})?) # new
                (?!
                  (?:
                    [a-zA-Z0-9-~,]
                    |
                    \),
                    |
                    \)\.
                  )
                )
```

Upon running

```
cmh:~$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml,itemize.yaml
```

we receive the output in Listing 412.

LISTING 412: `multiple-sentences6-mod3.tex` using Listing 407 and Listing 411

```
Consider the following:
\begin{itemize}
   \item firstly.
   \item secondly.
\end{itemize}
```

Notice that the sentence has received indentation, and that the `itemize` code block has been found and indented correctly. ■

U: 2022-04-04

Text wrapping when using the `oneSentencePerLine` routine determines if it will remove line breaks while text wrapping, from the value of `removeSentenceLineBreaks`.

6.2.8 oneSentencePerLine: text wrapping and indenting sentences, when before/after

N: 2023-01-01

The text wrapping routine operates, by default, `before` the code blocks have been found, but this can be changed to `after`:

- `before` means it is likely that the columns of wrapped text may *exceed* the value specified in `columns`;
- `after` means it columns of wrapped text should *not* exceed the value specified in `columns`.

We demonstrate this in the following examples. See also Section 6.1.7.

example 117 Let's begin with the file in Listing 413.

LISTING 413: `multiple-sentences8.tex`

```
This paragraph
has line breaks throughout its paragraph;
we would like to combine
the textwrapping
and paragraph removal routine.
\begin{myenv}
This paragraph
has line breaks throughout its paragraph;
we would like to combine
the textwrapping
and paragraph removal routine.
\end{myenv}
```

Using the settings given in Listing 415 and running the command

```
cmh:~$ latexindent.pl multiple-sentences8 -o=+-mod1.tex -l=sentence-wrap2 -m
```

gives the output given in Listing 414.

LISTING 414: `multiple-sentences8-mod1.tex`

```
This paragraph has line breaks
throughout its paragraph; we would
like to combine the textwrapping
and paragraph removal routine.
\begin{myenv}
   This paragraph has line breaks
   throughout its paragraph; we would
   like to combine the textwrapping
   and paragraph removal routine.
\end{myenv}
----|----|----|----|----|----|----|----|
   5   10   15   20   25   30   35   40
```

LISTING 415: `sentence-wrap2.yaml` -m

```
defaultIndent: '    '
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        textWrapSentences: 1
    textWrapOptions:
        columns: 35
        when: before   # <!-------
```

We note that, in Listing 414, that the wrapped text has *exceeded* the specified value of `columns` (35) given in Listing 415. We can affect this by changing `when`; we explore this next. ■

example 118 We continue working with Listing 413.

Using the settings given in Listing 417 and running the command

```
cmh:~$ latexindent.pl multiple-sentences8.tex -o=+-mod2.tex -l=sentence-wrap3 -m
```

gives the output given in Listing 416. ■

LISTING 416: `multiple-sentences8-mod2.tex`

```
This paragraph has line breaks
throughout its paragraph; we would
like to combine the textwrapping
and paragraph removal routine.
\begin{myenv}
   This paragraph has line breaks
   throughout its paragraph; we
   would like to combine the
   textwrapping and paragraph
   removal routine.
\end{myenv}
----|----|----|----|----|----|----|----|
   5   10   15   20   25   30   35   40
```

LISTING 417: `sentence-wrap3.yaml` -m

```
defaultIndent: '    '
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        textWrapSentences: 1
    textWrapOptions:
        columns: 35
        when: after  # <!-------
```

We note that, in Listing 416, that the wrapped text has *obeyed* the specified value of `columns` (35) given in Listing 417.

■

6.2.9 oneSentencePerLine: text wrapping sentences and comments

We demonstrate the one sentence per line routine with respect to text wrapping *comments*. See also Section 6.1.8.

example 119 Let's begin with the file in Listing 418.

LISTING 418: `multiple-sentences9.tex`

```
This paragraph% first comment
has line breaks throughout its paragraph;% second comment
we would like to combine% third comment
the textwrapping% fourth comment
and paragraph removal routine. % fifth comment
```

Using the settings given in Listing 420 and running the command

```
cmh:~$ latexindent.pl multiple-sentences9 -o=+-mod1.tex -l=sentence-wrap4 -m
```

gives the output given in Listing 419.

LISTING 419: `multiple-sentences9-mod1.tex`

```
This paragraph has line breaks
throughout its paragraph; we would
like to combine the textwrapping
and paragraph removal routine.
% first comment second comment
% third comment fourth comment
% fifth comment
----|----|----|----|----|----|----|----|
   5   10   15   20   25   30   35   40
```

LISTING 420: `sentence-wrap4.yaml` -m

```
defaultIndent: '    '
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        textWrapSentences: 1
    textWrapOptions:
        columns: 35
        comments:
          wrap: 1  #<!-------
```

We note that, in Listing 419, that the sentences have been wrapped, and so too have the comments because of the annotated line in Listing 420.

■

6.3 Poly-switches

U: 2017-08-21

Every other field in the `modifyLineBreaks` field uses poly-switches, and can take one of the following integer values:

−1 *remove mode*: line breaks before or after the <*part of thing*> can be removed (assuming that `preserveBlankLines` is set to 0);

0 *off mode*: line breaks will not be modified for the <*part of thing*> under consideration;

1 *add mode*: a line break will be added before or after the <*part of thing*> under consideration, assuming that there is not already a line break before or after the <*part of thing*>;

2 *comment then add mode*: a comment symbol will be added, followed by a line break before or after the <*part of thing*> under consideration, assuming that there is not already a comment and line break before or after the <*part of thing*>;

N: 2017-08-21

3 *add then blank line mode*: a line break will be added before or after the <*part of thing*> under consideration, assuming that there is not already a line break before or after the <*part of thing*>, followed by a blank line;

N: 2019-07-13

4 *add blank line mode*; a blank line will be added before or after the <*part of thing*> under consideration, even if the <*part of thing*> is already on its own line.

In the above, <*part of thing*> refers to either the *begin statement*, *body* or *end statement* of the code blocks detailed in Table 2 on page 56. All poly-switches are *off* by default; `latexindent.pl` searches first of all for per-name settings, and then followed by global per-thing settings.

6.3.1 Poly-switches for environments

We start by viewing a snippet of `defaultSettings.yaml` in Listing 421; note that it contains *global* settings (immediately after the `environments` field) and that *per-name* settings are also allowed – in the case of Listing 421, settings for `equation*` have been specified for demonstration. Note that all poly-switches are *off* (set to 0) by default.

Listing 421: `environments` -m

```
environments:
  BeginStartsOnOwnLine: 0             # -1,0,1,2,3,4
  BodyStartsOnOwnLine: 0              # -1,0,1,2,3,4
  EndStartsOnOwnLine: 0               # -1,0,1,2,3,4
  EndFinishesWithLineBreak: 0         # -1,0,1,2,3,4
  # equation*:
  #     BeginStartsOnOwnLine: 0         # -1,0,1,2,3,4
  #     BodyStartsOnOwnLine: 0          # -1,0,1,2,3,4
  #     EndStartsOnOwnLine: 0           # -1,0,1,2,3,4
  #     EndFinishesWithLineBreak: 0     # -1,0,1,2,3,4
```

Let's begin with the simple example given in Listing 422; note that we have annotated key parts of the file using ♠, ♡, ◇ and ♣, these will be related to fields specified in Listing 421.

Listing 422: `env-mlb1.tex`

```
before words♠ \begin{myenv}♡body of myenv◇\end{myenv}♣ after words
```

6.3.1.1 Adding line breaks: BeginStartsOnOwnLine and BodyStartsOnOwnLine

example 120 Let's explore `BeginStartsOnOwnLine` and `BodyStartsOnOwnLine` in Listings 423 and 424, and in particular, let's allow each of them in turn to take a value of 1.

Listing 423: `env-mlb1.yaml` -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 1
```

Listing 424: `env-mlb2.yaml` -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 1
```

After running the following commands,

■

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb1.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb2.yaml
```

the output is as in Listings 425 and 426 respectively.

Listing 425: `env-mlb.tex` using Listing 423

```
before words
\begin{myenv}body of myenv\end{myenv} after words
```

Listing 426: `env-mlb.tex` using Listing 424

```
before words \begin{myenv}
   body of myenv\end{myenv} after words
```

There are a couple of points to note:

- in Listing 425 a line break has been added at the point denoted by ♠ in Listing 422; no other line breaks have been changed;
- in Listing 426 a line break has been added at the point denoted by ♡ in Listing 422; furthermore, note that the *body* of `myenv` has received the appropriate (default) indentation. ■

example 121 Let's now change each of the `1` values in Listings 423 and 424 so that they are `2` and save them into `env-mlb3.yaml` and `env-mlb4.yaml` respectively (see Listings 427 and 428).

Listing 427: `env-mlb3.yaml` -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 2
```

Listing 428: `env-mlb4.yaml` -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 2
```

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb3.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb4.yaml
```

we obtain Listings 429 and 430.

Listing 429: `env-mlb.tex` using Listing 427

```
before words%
\begin{myenv}body of myenv\end{myenv} after words
```

Listing 430: `env-mlb.tex` using Listing 428

```
before words \begin{myenv}%
   body of myenv\end{myenv} after words
```

Note that line breaks have been added as in Listings 425 and 426, but this time a comment symbol has been added before adding the line break; in both cases, trailing horizontal space has been stripped before doing so. ■

example 122 Let's now change each of the `1` values in Listings 423 and 424 so that they are `3` and save them into `env-mlb5.yaml` and `env-mlb6.yaml` respectively (see Listings 431 and 432).

N: 2017-08-21

Listing 431: `env-mlb5.yaml` -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 3
```

Listing 432: `env-mlb6.yaml` -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 3
```

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb5.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb6.yaml
```

we obtain Listings 433 and 434. ■

LISTING 433: env-mlb.tex using Listing 431

```
before words

\begin{myenv}body of myenv\end{myenv} after words
```

LISTING 434: env-mlb.tex using Listing 432

```
before words \begin{myenv}

   body of myenv\end{myenv} after words
```

Note that line breaks have been added as in Listings 425 and 426, but this time a *blank line* has been added after adding the line break.

example 123 Let's now change each of the 1 values in Listings 431 and 432 so that they are 4 and save them into `env-beg4.yaml` and `env-body4.yaml` respectively (see Listings 435 and 436).

N: 2019-07-13

LISTING 435: env-beg4.yaml -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 4
```

LISTING 436: env-body4.yaml -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 4
```

We will demonstrate this poly-switch value using the code in Listing 437.

LISTING 437: env-mlb1.tex

```
before words
\begin{myenv}
body of myenv
\end{myenv}
after words
```

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb1.tex -l env-beg4.yaml
cmh:~$ latexindent.pl -m env-mlb.1tex -l env-body4.yaml
```

then we receive the respective outputs in Listings 438 and 439.

LISTING 438: env-mlb1.tex using Listing 435

```
before words

\begin{myenv}
   body of myenv
\end{myenv}
after words
```

LISTING 439: env-mlb1.tex using Listing 436

```
before words
\begin{myenv}

   body of myenv
\end{myenv}
after words
```

We note in particular that, by design, for this value of the poly-switches:

1. in Listing 438 a blank line has been inserted before the `\begin` statement, even though the `\begin` statement was already on its own line;
2. in Listing 439 a blank line has been inserted before the beginning of the *body*, even though it already began on its own line.

6.3.1.2 Adding line breaks: EndStartsOnOwnLine and EndFinishesWithLineBreak

example 124 Let's explore `EndStartsOnOwnLine` and `EndFinishesWithLineBreak` in Listings 440 and 441, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 440: env-mlb7.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 1
```

LISTING 441: env-mlb8.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 1
```

After running the following commands,

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb7.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb8.yaml
```

the output is as in Listings 442 and 443.

LISTING 442: env-mlb.tex using Listing 440

```
before words \begin{myenv}body
    of myenv
\end{myenv} after words
```

LISTING 443: env-mlb.tex using Listing 441

```
before words \begin{myenv}body
    of myenv\end{myenv}
after words
```

There are a couple of points to note:

- in Listing 442 a line break has been added at the point denoted by ♢ in Listing 422 on page 109; no other line breaks have been changed and the `\end{myenv}` statement has *not* received indentation (as intended);
- in Listing 443 a line break has been added at the point denoted by ♣ in Listing 422 on page 109. ■

example 125 Let's now change each of the 1 values in Listings 440 and 441 so that they are 2 and save them into `env-mlb9.yaml` and `env-mlb10.yaml` respectively (see Listings 444 and 445).

LISTING 444: env-mlb9.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 2
```

LISTING 445: env-mlb10.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 2
```

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb9.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb10.yaml
```

we obtain Listings 446 and 447.

LISTING 446: env-mlb.tex using Listing 444

```
before words \begin{myenv}body of myenv%
\end{myenv} after words
```

LISTING 447: env-mlb.tex using Listing 445

```
before words \begin{myenv}body of myenv\end{myenv}%
after words
```

Note that line breaks have been added as in Listings 442 and 443, but this time a comment symbol has been added before adding the line break; in both cases, trailing horizontal space has been stripped before doing so. ■

example 126 Let's now change each of the 1 values in Listings 440 and 441 so that they are 3 and save them into `env-mlb11.yaml` and `env-mlb12.yaml` respectively (see Listings 448 and 449). ■

N: 2017-08-21

LISTING 448: `env-mlb11.yaml` -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 3
```

LISTING 449: `env-mlb12.yaml` -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 3
```

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb11.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb12.yaml
```

we obtain Listings 450 and 451.

LISTING 450: `env-mlb.tex` using Listing 448

```
before words \begin{myenv}body of myenv

\end{myenv} after words
```

LISTING 451: `env-mlb.tex` using Listing 449

```
before words \begin{myenv}body of myenv\end{myenv}

after words
```

Note that line breaks have been added as in Listings 442 and 443, and that a *blank line* has been added after the line break. ■

example 127 Let's now change each of the 1 values in Listings 448 and 449 so that they are 4 and save them into `env-end4.yaml` and `env-end-f4.yaml` respectively (see Listings 452 and 453).

N: 2019-07-13

LISTING 452: `env-end4.yaml` -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 4
```

LISTING 453: `env-end-f4.yaml` -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 4
```

We will demonstrate this poly-switch value using the code from Listing 437 on page 111.

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb1.tex -l env-end4.yaml
cmh:~$ latexindent.pl -m env-mlb.1tex -l env-end-f4.yaml
```

then we receive the respective outputs in Listings 454 and 455.

LISTING 454: `env-mlb1.tex` using Listing 452

```
before words
\begin{myenv}
   body of myenv

\end{myenv}
after words
```

LISTING 455: `env-mlb1.tex` using Listing 453

```
before words
\begin{myenv}
   body of myenv
\end{myenv}

after words
```

We note in particular that, by design, for this value of the poly-switches:

1. in Listing 454 a blank line has been inserted before the `\end` statement, even though the `\end` statement was already on its own line;
2. in Listing 455 a blank line has been inserted after the `\end` statement, even though it already began on its own line. ■

6.3.1.3 poly-switches 1, 2, and 3 only add line breaks when necessary

If you ask `latexindent.pl` to add a line break (possibly with a comment) using a poly-switch value of 1 (or 2 or 3), it will only do so if necessary.

example 128 For example, if you process the file in Listing 456 using poly-switch values of 1, 2, or 3, it will be left unchanged.

LISTING 456: env-mlb2.tex

```
before words
\begin{myenv}
  body of myenv
\end{myenv}
after words
```

LISTING 457: env-mlb3.tex

```
before words
\begin{myenv}  %
  body of myenv%
\end{myenv}%
after words
```

■

Setting the poly-switches to a value of 4 instructs `latexindent.pl` to add a line break even if the *<part of thing>* is already on its own line; see Listings 438 and 439 and Listings 454 and 455.

example 129 In contrast, the output from processing the file in Listing 457 will vary depending on the poly-switches used; in Listing 458 you'll see that the comment symbol after the `\begin{myenv}` has been moved to the next line, as `BodyStartsOnOwnLine` is set to 1. In Listing 459 you'll see that the comment has been accounted for correctly because `BodyStartsOnOwnLine` has been set to 2, and the comment symbol has *not* been moved to its own line. You're encouraged to experiment with Listing 457 and by setting the other poly-switches considered so far to 2 in turn.

LISTING 458: env-mlb3.tex using Listing 424 on page 109

```
before words
\begin{myenv}
   %
   body of myenv%
\end{myenv}%
after words
```

LISTING 459: env-mlb3.tex using Listing 428 on page 110

```
before words
\begin{myenv}  %
   body of myenv%
\end{myenv}%
after words
```

■

The details of the discussion in this section have concerned *global* poly-switches in the `environments` field; each switch can also be specified on a *per-name* basis, which would take priority over the global values; with reference to Listing 421 on page 109, an example is shown for the `equation*` environment.

6.3.1.4 Removing line breaks (poly-switches set to −1)

Setting poly-switches to −1 tells `latexindent.pl` to remove line breaks of the *<part of the thing>*, if necessary.

example 130 We will consider the example code given in Listing 460, noting in particular the positions of the line break highlighters, ♠, ♡, ♢ and ♣, together with the associated YAML files in Listings 461 to 464.

■

LISTING 460: env-mlb4.tex

```
before words♠
\begin{myenv}♡
body of myenv♢
\end{myenv}♣
after words
```

LISTING 461: env-mlb13.yaml -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: -1
```

LISTING 462: env-mlb14.yaml -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: -1
```

LISTING 463: env-mlb15.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: -1
```

LISTING 464: env-mlb16.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak:
    -1
```

After running the commands

```
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb13.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb14.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb15.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb16.yaml
```

we obtain the respective output in Listings 465 to 468.

LISTING 465: env-mlb4.tex using Listing 461

```
before words\begin{myenv}
    body of myenv
\end{myenv}
after words
```

LISTING 466: env-mlb4.tex using Listing 462

```
before words
\begin{myenv}body of myenv
\end{myenv}
after words
```

LISTING 467: env-mlb4.tex using Listing 463

```
before words
\begin{myenv}
    body of myenv\end{myenv}
after words
```

LISTING 468: env-mlb4.tex using Listing 464

```
before words
\begin{myenv}
    body of myenv
\end{myenv}after words
```

Notice that in:

- Listing 465 the line break denoted by ♠ in Listing 460 has been removed;
- Listing 466 the line break denoted by ♡ in Listing 460 has been removed;
- Listing 467 the line break denoted by ♢ in Listing 460 has been removed;
- Listing 468 the line break denoted by ♣ in Listing 460 has been removed.

We examined each of these cases separately for clarity of explanation, but you can combine all of the YAML settings in Listings 461 to 464 into one file; alternatively, you could tell `latexindent.pl` to load them all by using the following command, for example

```
cmh:~$
    latexindent.pl -m env-mlb4.tex -l env-mlb13.yaml,env-mlb14.yaml,env-mlb15.yaml,env-mlb16.yaml
```

which gives the output in Listing 422 on page 109. ■

6.3.1.5 About trailing horizontal space

Recall that on page 33 we discussed the YAML field `removeTrailingWhitespace`, and that it has two (binary) switches to determine if horizontal space should be removed `beforeProcessing` and `afterProcessing`. The `beforeProcessing` is particularly relevant when considering the `-m` switch.

example 131 We consider the file shown in Listing 469, which highlights trailing spaces.

LISTING 469: `env-mlb5.tex`

```
before␣words␣␣␣♠
\begin{myenv}␣␣␣␣␣␣␣␣␣␣␣♡
body␣of␣myenv␣␣␣␣␣␣♢
\end{myenv}␣␣␣␣␣♣
after␣words
```

LISTING 470: `removeTWS-before.yaml`

```
removeTrailingWhitespace:
    beforeProcessing: 1
```

The output from the following commands

```
cmh:~$ latexindent.pl -m env-mlb5.tex -l env-mlb13,env-mlb14,env-mlb15,env-mlb16
cmh:~$ latexindent.pl -m env-mlb5.tex -l env-mlb13,env-mlb14,env-mlb15,env-mlb16,removeTWS-before
```

is shown, respectively, in Listings 471 and 472; note that the trailing horizontal white space has been preserved (by default) in Listing 471, while in Listing 472, it has been removed using the switch specified in Listing 470.

LISTING 471: `env-mlb5.tex` using Listings 465 to 468

```
before␣words␣␣␣\begin{myenv}␣␣␣␣␣␣␣␣␣␣␣body␣of␣myenv␣␣␣␣␣␣\end{myenv}␣␣␣␣␣after␣words
```

LISTING 472: `env-mlb5.tex` using Listings 465 to 468 *and* Listing 470

```
before␣words\begin{myenv}body␣of␣myenv\end{myenv}after␣words
```

■

6.3.1.6 poly-switch line break removal and blank lines

example 132 Now let's consider the file in Listing 473, which contains blank lines.

LISTING 473: `env-mlb6.tex`

```
before words♠


\begin{myenv}♡


body of myenv♢


\end{myenv}♣

after words
```

LISTING 474: `UnpreserveBlankLines.yaml` `-m`

```
modifyLineBreaks:
    preserveBlankLines: 0
```

Upon running the following commands ■

```
cmh:~$ latexindent.pl -m env-mlb6.tex -l env-mlb13,env-mlb14,env-mlb15,env-mlb16
cmh:~$
    latexindent.pl -m env-mlb6.tex -l env-mlb13,env-mlb14,env-mlb15,env-mlb16,UnpreserveBlankLines
```

we receive the respective outputs in Listings 475 and 476. In Listing 475 we see that the multiple blank lines have each been condensed into one blank line, but that blank lines have *not* been removed by the poly-switches – this is because, by default, `preserveBlankLines` is set to `1`. By contrast, in Listing 476, we have allowed the poly-switches to remove blank lines because, in Listing 474, we have set `preserveBlankLines` to `0`.

LISTING 475: `env-mlb6.tex` using Listings 465 to 468

```
before words

\begin{myenv}

   body of myenv

\end{myenv}

after words
```

LISTING 476: `env-mlb6.tex` using Listings 465 to 468 *and* Listing 474

```
before words\begin{myenv}body of myenv\end{myenv}after words
```

■

example 133 We can explore this further using the blank-line poly-switch value of 3; let's use the file given in Listing 477.

LISTING 477: `env-mlb7.tex`

```
\begin{one} one text \end{one} \begin{two} two text \end{two}
```

Upon running the following commands

```
cmh:~$ latexindent.pl -m env-mlb7.tex -l env-mlb12.yaml,env-mlb13.yaml
cmh:~$
    latexindent.pl -m env-mlb7.tex -l env-mlb13,env-mlb14,UnpreserveBlankLines
```

we receive the outputs given in Listings 478 and 479.

LISTING 478: `env-mlb7-preserve.tex`

```
\begin{one} one text \end{one}

\begin{two} two text \end{two}
```

LISTING 479: `env-mlb7-no-preserve.tex`

```
\begin{one} one text \end{one} \begin{two} two text \end{two}
```

Notice that in:

- Listing 478 that `\end{one}` has added a blank line, because of the value of `EndFinishesWithLineBreak` in Listing 449 on page 113, and even though the line break ahead of `\begin{two}` should have been removed (because of `BeginStartsOnOwnLine` in Listing 461 on page 115), the blank line has been preserved by default;
- Listing 479, by contrast, has had the additional line-break removed, because of the settings in Listing 474.

■

6.3.2 Poly-switches for double backslash

N: 2019-07-13

With reference to `lookForAlignDelims` (see Listing 58 on page 33) you can specify poly-switches to dictate the line-break behaviour of double backslashes in environments (Listing 60 on page 34), commands (Listing 94 on page 40), or special code blocks (Listing 139 on page 48). [6]

Consider the code given in Listing 480.

LISTING 480: `tabular3.tex`

```
\begin{tabular}{cc}
 1 & 2 ★\\□ 3 & 4 ★\\□
\end{tabular}
```

Referencing Listing 480:

- DBS stands for *double backslash*;
- line breaks ahead of the double backslash are annotated by ★, and are controlled by `DBSStartsOnOwnLine`;
- line breaks after the double backslash are annotated by □, and are controlled by `DBSFinishesWithLineBreak`.

Let's explore each of these in turn.

6.3.2.1 Double backslash starts on own line

example 134 We explore `DBSStartsOnOwnLine` (★ in Listing 480); starting with the code in Listing 480, together with the YAML files given in Listing 482 and Listing 484 and running the following commands

```
cmh:~$ latexindent.pl -m tabular3.tex -l DBS1.yaml
cmh:~$ latexindent.pl -m tabular3.tex -l DBS2.yaml
```

then we receive the respective output given in Listing 481 and Listing 483.

LISTING 481: `tabular3.tex` using Listing 482

```
\begin{tabular}{cc}
   1 & 2
   \\ 3 & 4
   \\
\end{tabular}
```

LISTING 482: `DBS1.yaml` -m

```
modifyLineBreaks:
    environments:
        DBSStartsOnOwnLine: 1
```

LISTING 483: `tabular3.tex` using Listing 484

```
\begin{tabular}{cc}
   1 & 2 %
   \\ 3 & 4%
   \\
\end{tabular}
```

LISTING 484: `DBS2.yaml` -m

```
modifyLineBreaks:
    environments:
        tabular:
            DBSStartsOnOwnLine: 2
```

We note that

- Listing 482 specifies `DBSStartsOnOwnLine` for *every* environment (that is within `lookForAlignDelims`, Listing 61 on page 34); the double backslashes from Listing 480 have been moved to their own line in Listing 481;
- Listing 484 specifies `DBSStartsOnOwnLine` on a *per-name* basis for `tabular` (that is within `lookForAlignDelims`, Listing 61 on page 34); the double backslashes from Listing 480 have been moved to their own line in Listing 483, having added comment symbols before moving them.

■

U: 2023-01-01

[6] There is no longer any need for the code block to be specified within `lookForAlignDelims` for DBS poly-switches to activate.

example 135 We can combine DBS poly-switches with, for example, the `alignContentAfterDoubleBackSlash` in Section 5.5.6 on page 45.

For example, starting with the file Listing 485, and using the settings in Listings 130 and 132 on page 46 and running

```
cmh:~$ latexindent.pl -s -m -l alignContentAfterDBS1.yaml,DBS1.yaml tabular6.tex -o=+-mod1
cmh:~$ latexindent.pl -s -m -l alignContentAfterDBS2.yaml,DBS1.yaml tabular6.tex -o=+-mod2
```

gives the respective outputs shown in Listings 486 and 487.

LISTING 485: `tabular6.tex`

```
\begin{tabular}{cc}
 1&22\\333&4444\\55555&666666
\end{tabular}
```

LISTING 486: `tabular6-mod1.tex`

```
\begin{tabular}{cc}
      1     & 22
   \\ 333   & 4444
   \\ 55555 & 666666
\end{tabular}
```

LISTING 487: `tabular6-mod2.tex`

```
\begin{tabular}{cc}
        1     & 22
   \\   333   & 4444
   \\   55555 & 666666
\end{tabular}
```

We note that:

- in Listing 486 the content *after* the double back slash has been aligned;
- in Listing 487 we see that 3 spaces have been added after the double back slash. ■

6.3.2.2 Double backslash finishes with line break

example 136 Let's now explore `DBSFinishesWithLineBreak` (☐ in Listing 480); starting with the code in Listing 480, together with the YAML files given in Listing 489 and Listing 491 and running the following commands

```
cmh:~$ latexindent.pl -m tabular3.tex -l DBS3.yaml
cmh:~$ latexindent.pl -m tabular3.tex -l DBS4.yaml
```

then we receive the respective output given in Listing 488 and Listing 490.

LISTING 488: `tabular3.tex` using Listing 489

```
\begin{tabular}{cc}
   1 & 2 \\
   3 & 4 \\
\end{tabular}
```

LISTING 489: `DBS3.yaml` -m

```
modifyLineBreaks:
    environments:
        DBSFinishesWithLineBreak: 1
```

LISTING 490: `tabular3.tex` using Listing 491

```
\begin{tabular}{cc}
   1 & 2 \\%
   3 & 4 \\
\end{tabular}
```

LISTING 491: `DBS4.yaml` -m

```
modifyLineBreaks:
    environments:
        tabular:
            DBSFinishesWithLineBreak:
    2
```

We note that

- Listing 489 specifies `DBSFinishesWithLineBreak` for *every* environment (that is within `lookForAlignDelims`, Listing 61 on page 34); the code following the double backslashes from Listing 480 has been moved to their own line in Listing 488;
- Listing 491 specifies `DBSFinishesWithLineBreak` on a *per-name* basis for `tabular` (that is within `lookForAlignDelims`, Listing 61 on page 34); the first double backslashes from Listing 480 have moved code following them to their own line in Listing 490, having added comment symbols before moving them; the final double backslashes have *not* added a line ■

break as they are at the end of the body within the code block.

6.3.2.3 Double backslash poly-switches for specialBeginEnd

example 137 Let’s explore the double backslash poly-switches for code blocks within `specialBeginEnd` code blocks (Listing 137 on page 47); we begin with the code within Listing 492.

LISTING 492: `special4.tex`

```
\< a& =b  \\ & =c\\ & =d\\ & =e \>
```

Upon using the YAML settings in Listing 494, and running the command

```
cmh:~$ latexindent.pl -m special4.tex -l DBS5.yaml
```

then we receive the output given in Listing 493.

LISTING 493: `special4.tex` using Listing 494

```
\<
   a & =b \\
     & =c \\
     & =d \\
     & =e %
\>
```

LISTING 494: `DBS5.yaml`

-m

```
specialBeginEnd:
    cmhMath:
        lookForThis: 1
        begin: ’\\<’
        end: ’\\>’
lookForAlignDelims:
    cmhMath: 1
modifyLineBreaks:
    specialBeginEnd:
        cmhMath:
            DBSFinishesWithLineBreak: 1
            SpecialBodyStartsOnOwnLine: 1
            SpecialEndStartsOnOwnLine: 2
```

There are a few things to note:

- in Listing 494 we have specified `cmhMath` within `lookForAlignDelims`; without this, the double backslash poly-switches would be ignored for this code block;
- the `DBSFinishesWithLineBreak` poly-switch has controlled the line breaks following the double backslashes;
- the `SpecialEndStartsOnOwnLine` poly-switch has controlled the addition of a comment symbol, followed by a line break, as it is set to a value of 2.

6.3.2.4 Double backslash poly-switches for optional and mandatory arguments

For clarity, we provide a demonstration of controlling the double backslash poly-switches for optional and mandatory arguments.

example 138 We use with the code in Listing 495.

LISTING 495: `mycommand2.tex`

```
\mycommand [
   1&2   &3\\ 4&5&6]{
7&8   &9\\ 10&11&12
}
```

Upon using the YAML settings in Listings 497 and 499, and running the command

```
cmh:~$ latexindent.pl -m mycommand2.tex -l DBS6.yaml
cmh:~$ latexindent.pl -m mycommand2.tex -l DBS7.yaml
```

then we receive the output given in Listings 496 and 498.

LISTING 496: `mycommand2.tex` using Listing 497

```
\mycommand [
   1 & 2 & 3 %
   \\%
   4 & 5 & 6]{
   7 & 8 & 9 \\ 10&11&12
}
```

LISTING 497: `DBS6.yaml` -m

```
lookForAlignDelims:
    mycommand: 1
modifyLineBreaks:
    optionalArguments:
        DBSStartsOnOwnLine: 2
        DBSFinishesWithLineBreak: 2
```

LISTING 498: `mycommand2.tex` using Listing 499

```
\mycommand [
   1&2   &3\\ 4&5&6]{
   7  & 8  & 9  %
   \\%
   10 & 11 & 12
}
```

LISTING 499: `DBS7.yaml` -m

```
lookForAlignDelims:
    mycommand: 1
modifyLineBreaks:
    mandatoryArguments:
        DBSStartsOnOwnLine: 2
        DBSFinishesWithLineBreak: 2
```

6.3.2.5 Double backslash optional square brackets

The pattern matching for the double backslash will also, optionally, allow trailing square brackets that contain a measurement of vertical spacing, for example `\\[3pt]`.

example 139 For example, beginning with the code in Listing 500

LISTING 500: `pmatrix3.tex`

```
\begin{pmatrix}
1 & 2 \\[2pt] 3 & 4 \\ [ 3 ex] 5&6\\[    4  pt   ] 7 & 8
\end{pmatrix}
```

and running the following command, using Listing 489,

```
cmh:~$ latexindent.pl -m pmatrix3.tex -l DBS3.yaml
```

then we receive the output given in Listing 501.

LISTING 501: `pmatrix3.tex` using Listing 489

```
\begin{pmatrix}
   1 & 2 \\[2pt]
   3 & 4 \\ [ 3 ex]
   5 & 6 \\[    4  pt   ]
   7 & 8
\end{pmatrix}
```

You can customise the pattern for the double backslash by exploring the *fine tuning* field detailed in Listing 566 on page 144.

6.3.3 Poly-switches for other code blocks

Rather than repeat the examples shown for the environment code blocks (in Section 6.3.1 on page 109), we choose to detail the poly-switches for all other code blocks in Table 3; note that each and every one of these poly-switches is *off by default*, i.e, set to `0`.

U: 2019-05-05

Note also that, by design, line breaks involving, `filecontents` and ‘comment-marked’ code blocks (Listing 95 on page 41) can *not* be modified using `latexindent.pl`. However, there are two poly-switches available for `verbatim` code blocks: environments (Listing 37 on page 29), commands (Listing 38 on page 29) and `specialBeginEnd` (Listing 154 on page 51).

TABLE 3: Poly-switch mappings for all code-block types

N: 2018-04-27

N: 2019-07-13

N: 2019-07-13

N: 2018-04-27

Code block	Sample		Poly-switch mapping
environment	`before words`♠ `\begin{myenv}`♡ `body of myenv`♢ `\end{myenv}`♣ `after words`	♠ ♡ ♢ ♣	BeginStartsOnOwnLine BodyStartsOnOwnLine EndStartsOnOwnLine EndFinishesWithLineBreak
ifelsefi	`before words`♠ `\if...`♡ `body of if/or statement`▲ `\or`▼ `body of if/or statement`★ `\else`□ `body of else statement`♢ `\fi`♣ `after words`	♠ ♡ ▲ ▼ ★ □ ♢ ♣	IfStartsOnOwnLine BodyStartsOnOwnLine OrStartsOnOwnLine OrFinishesWithLineBreak ElseStartsOnOwnLine ElseFinishesWithLineBreak FiStartsOnOwnLine FiFinishesWithLineBreak

optionalArguments	`...`♠
	`[`♡
	`value before comma`★`,`
	□
	`end of body of opt arg`♢
	`]`♣
	`...`
		♠
		♡
		★
		□
		♢
		♣
			LSqBStartsOnOwnLine[7]
			OptArgBodyStartsOnOwnLine
			CommaStartsOnOwnLine
			CommaFinishesWithLineBreak
			RSqBStartsOnOwnLine
			RSqBFinishesWithLineBreak

mandatoryArguments	`...`♠
	`{`♡
	`value before comma`★`,`
	□
	`end of body of mand arg`♢
	`}`♣
	`...`
		♠
		♡
		★
		□
		♢
		♣
			LCuBStartsOnOwnLine[8]
			MandArgBodyStartsOnOwnLine
			CommaStartsOnOwnLine
			CommaFinishesWithLineBreak
			RCuBStartsOnOwnLine
			RCuBFinishesWithLineBreak

commands	`before words`♠ `\mycommand`♡ ⟨*arguments*⟩	♠ ♡	CommandStartsOnOwnLine CommandNameFinishesWithLineBr
namedGroupingBracesBrackets	before words♠ myname♡ ⟨*braces/brackets*⟩	♠ ♡	NameStartsOnOwnLine NameFinishesWithLineBreak
keyEqualsValuesBracesBrackets	before words♠ key•=♡ ⟨*braces/brackets*⟩	♠ • ♡	KeyStartsOnOwnLine EqualsStartsOnOwnLine EqualsFinishesWithLineBreak
items	before words♠ `\item`♡ `...`	♠ ♡	ItemStartsOnOwnLine ItemFinishesWithLineBreak
specialBeginEnd	before words♠ `\[`♡ `body of special/middle`★ `\middle`□ body of special/middle ♢ `\]`♣ after words	♠ ♡ ★ □ ♢ ♣	SpecialBeginStartsOnOwnLine SpecialBodyStartsOnOwnLine SpecialMiddleStartsOnOwnLine SpecialMiddleFinishesWithLineBrea SpecialEndStartsOnOwnLine SpecialEndFinishesWithLineBreak
verbatim	before words♠`\begin{verbatim}`	♠	VerbatimBeginStartsOnOwnLine

---

[7] LSqB stands for Left Square Bracket

[8] LCuB stands for Left Curly Brace

N: 2019-05-05

body of verbatim `\end{verbatim}`♣ after words	♣	VerbatimEndFinishesWithLineBreak

6.3.4 Partnering BodyStartsOnOwnLine with argument-based poly-switches

Some poly-switches need to be partnered together; in particular, when line breaks involving the *first* argument of a code block need to be accounted for using both `BodyStartsOnOwnLine` (or its equivalent, see Table 3 on the preceding page) and `LCuBStartsOnOwnLine` for mandatory arguments, and `LSqBStartsOnOwnLine` for optional arguments.

example 140 Let's begin with the code in Listing 502 and the YAML settings in Listing 504; with reference to Table 3 on the previous page, the key `CommandNameFinishesWithLineBreak` is an alias for `BodyStartsOnOwnLine`.

LISTING 502: `mycommand1.tex`

```
\mycommand
{
mand arg text
mand arg text}
{
mand arg text
mand arg text}
```

Upon running the command

```
cmh:~$ latexindent.pl -m -l=mycom-mlb1.yaml mycommand1.tex
```

we obtain Listing 503; note that the *second* mandatory argument beginning brace `{` has had its leading line break removed, but that the *first* brace has not.

LISTING 503: `mycommand1.tex` using Listing 504

```
\mycommand
{
   mand arg text
   mand arg text}{
   mand arg text
   mand arg text}
```

LISTING 504: `mycom-mlb1.yaml` -m

```
modifyLineBreaks:
    commands:
        CommandNameFinishesWithLineBreak: 0
    mandatoryArguments:
        LCuBStartsOnOwnLine: -1
```

■

example 141 Now let's change the YAML file so that it is as in Listing 506; upon running the command

```
cmh:~$ latexindent.pl -m -l=mycom-mlb2.yaml mycommand1.tex
```

we obtain Listing 505; both beginning braces `{` have had their leading line breaks removed.

LISTING 505: `mycommand1.tex` using Listing 506

```
\mycommand{
   mand arg text
   mand arg text}{
   mand arg text
   mand arg text}
```

LISTING 506: `mycom-mlb2.yaml` -m

```
modifyLineBreaks:
    commands:
        CommandNameFinishesWithLineBreak: -1
    mandatoryArguments:
        LCuBStartsOnOwnLine: -1
```

■

example 142 Now let's change the YAML file so that it is as in Listing 508; upon running the command ■

```
cmh:~$ latexindent.pl -m -l=mycom-mlb3.yaml mycommand1.tex
```

we obtain Listing 507.

LISTING 507: `mycommand1.tex` using Listing 508

```
\mycommand
{
   mand arg text
   mand arg text}
{
   mand arg text
   mand arg text}
```

LISTING 508: `mycom-mlb3.yaml` -m

```
modifyLineBreaks:
    commands:
        CommandNameFinishesWithLineBreak: -1
    mandatoryArguments:
        LCuBStartsOnOwnLine: 1
```

■

6.3.5 Conflicting poly-switches: sequential code blocks

It is very easy to have conflicting poly-switches.

example 143 We use the example from Listing 502 on the preceding page, and consider the YAML settings given in Listing 510. The output from running

```
cmh:~$ latexindent.pl -m -l=mycom-mlb4.yaml mycommand1.tex
```

is given in Listing 510.

LISTING 509: `mycommand1.tex` using Listing 510

```
\mycommand
{
   mand arg text
   mand arg text}{
   mand arg text
   mand arg text}
```

LISTING 510: `mycom-mlb4.yaml` -m

```
modifyLineBreaks:
    mandatoryArguments:
        LCuBStartsOnOwnLine: -1
        RCuBFinishesWithLineBreak: 1
```

Studying Listing 510, we see that the two poly-switches are at opposition with one another:

- on the one hand, `LCuBStartsOnOwnLine` should *not* start on its own line (as poly-switch is set to −1);
- on the other hand, `RCuBFinishesWithLineBreak` *should* finish with a line break.

So, which should win the conflict? As demonstrated in Listing 509, it is clear that `LCuBStartsOnOwnLine` won this conflict, and the reason is that *the second argument was processed after the first* – in general, the most recently-processed code block and associated poly-switch takes priority. ■

example 144 We can explore this further by considering the YAML settings in Listing 512; upon running the command

```
cmh:~$ latexindent.pl -m -l=mycom-mlb5.yaml mycommand1.tex
```

we obtain the output given in Listing 511. ■

LISTING 511: `mycommand1.tex` using Listing 512

```
\mycommand
{
   mand arg text
   mand arg text}
{
   mand arg text
   mand arg text}
```

LISTING 512: `mycom-mlb5.yaml` -m

```
modifyLineBreaks:
    mandatoryArguments:
        LCuBStartsOnOwnLine: 1
        RCuBFinishesWithLineBreak:
    -1
```

As previously, the most-recently-processed code block takes priority – as before, the second (i.e, *last*) argument.

Exploring this further, we consider the YAML settings in Listing 514, and run the command

```
cmh:~$ latexindent.pl -m -l=mycom-mlb6.yaml mycommand1.tex
```

which gives the output in Listing 513.

LISTING 513: `mycommand1.tex` using Listing 514

```
\mycommand
{
   mand arg text
   mand arg text}%
{
   mand arg text
   mand arg text}
```

LISTING 514: `mycom-mlb6.yaml` -m

```
modifyLineBreaks:
    mandatoryArguments:
        LCuBStartsOnOwnLine: 2
        RCuBFinishesWithLineBreak:
    -1
```

Note that a `%` *has* been added to the trailing first `}`; this is because:

- while processing the *first* argument, the trailing line break has been removed (`RCuBFinishesWithLineBreak` set to −1);
- while processing the *second* argument, `latexindent.pl` finds that it does *not* begin on its own line, and so because `LCuBStartsOnOwnLine` is set to 2, it adds a comment, followed by a line break.

■

6.3.6 Conflicting poly-switches: nested code blocks

example 145 Now let's consider an example when nested code blocks have conflicting poly-switches; we'll use the code in Listing 515, noting that it contains nested environments.

LISTING 515: `nested-env.tex`

```
\begin{one}
one text
\begin{two}
two text
\end{two}
\end{one}
```

Let's use the YAML settings given in Listing 517, which upon running the command

```
cmh:~$ latexindent.pl -m -l=nested-env-mlb1.yaml nested-env.tex
```

gives the output in Listing 516.

■

LISTING 516: `nested-env.tex` using Listing 517

```
\begin{one}
   one text
   \begin{two}
      two text\end{two}\end{one}
```

LISTING 517: `nested-env-mlb1.yaml` -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: -1
        EndFinishesWithLineBreak: 1
```

In Listing 516, let's first of all note that both environments have received the appropriate (default) indentation; secondly, note that the poly-switch `EndStartsOnOwnLine` appears to have won the conflict, as `\end{one}` has had its leading line break removed. ■

To understand it, let's talk about the three basic phases of `latexindent.pl`:

1. Phase 1: packing, in which code blocks are replaced with unique ids, working from *the inside to the outside*, and then sequentially – for example, in Listing 515, the `two` environment is found *before* the `one` environment; if the -m switch is active, then during this phase:
   - line breaks at the beginning of the `body` can be added (if `BodyStartsOnOwnLine` is 1 or 2) or removed (if `BodyStartsOnOwnLine` is −1);
   - line breaks at the end of the body can be added (if `EndStartsOnOwnLine` is 1 or 2) or removed (if `EndStartsOnOwnLine` is −1);
   - line breaks after the end statement can be added (if `EndFinishesWithLineBreak` is 1 or 2).
2. Phase 2: indentation, in which white space is added to the begin, body, and end statements;
3. Phase 3: unpacking, in which unique ids are replaced by their *indented* code blocks; if the -m switch is active, then during this phase,
   - line breaks before `begin` statements can be added or removed (depending upon `BeginStartsOnOwnLine`);
   - line breaks after *end* statements can be removed but *NOT* added (see `EndFinishesWithLineBreak`).

With reference to Listing 516, this means that during Phase 1:

- the `two` environment is found first, and the line break ahead of the `\end{two}` statement is removed because `EndStartsOnOwnLine` is set to −1. Importantly, because, *at this stage*, `\end{two}` *does* finish with a line break, `EndFinishesWithLineBreak` causes no action.
- next, the `one` environment is found; the line break ahead of `\end{one}` is removed because `EndStartsOnOwnLine` is set to −1.

The indentation is done in Phase 2; in Phase 3 *there is no option to add a line break after the `end` statements*. We can justify this by remembering that during Phase 3, the `one` environment will be found and processed first, followed by the `two` environment. If the `two` environment were to add a line break after the `\end{two}` statement, then `latexindent.pl` would have no way of knowing how much indentation to add to the subsequent text (in this case, `\end{one}`).

example 146 We can explore this further using the poly-switches in Listing 519; upon running the command

```
cmh:~$ latexindent.pl -m -l=nested-env-mlb2.yaml nested-env.tex
```

we obtain the output given in Listing 518. ■

LISTING 518: `nested-env.tex` using Listing 519

```
\begin{one}
   one text
   \begin{two}
      two text
   \end{two}\end{one}
```

LISTING 519: `nested-env-mlb2.yaml` -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 1
        EndFinishesWithLineBreak: -1
```

During Phase 1:

- the `two` environment is found first, and the line break ahead of the `\end{two}` statement is not changed because `EndStartsOnOwnLine` is set to 1. Importantly, because, *at this stage*, `\end{two}` *does* finish with a line break, `EndFinishesWithLineBreak` causes no action.
- next, the `one` environment is found; the line break ahead of `\end{one}` is already present, and no action is needed.

The indentation is done in Phase 2, and then in Phase 3, the `one` environment is found and processed first, followed by the `two` environment. *At this stage*, the `two` environment finds `EndFinishesWithLineBreak` is −1, so it removes the trailing line break; remember, at this point, `latexindent.pl` has completely finished with the `one` environment. ■

SECTION 7

The -r, -rv and -rr switches

N: 2019-07-13

You can instruct `latexindent.pl` to perform replacements/substitutions on your file by using any of the `-r`, `-rv` or `-rr` switches:

- the `-r` switch will perform indentation and replacements, not respecting verbatim code blocks;
- the `-rv` switch will perform indentation and replacements, and *will* respect verbatim code blocks;
- the `-rr` switch will *not* perform indentation, and will perform replacements not respecting verbatim code blocks.

We will demonstrate each of the `-r`, `-rv` and `-rr` switches, but a summary is given in Table 4.

TABLE 4: The replacement mode switches

switch	indentation?	respect verbatim?
`-r`	✔	✖
`-rv`	✔	✔
`-rr`	✖	✖

The default value of the `replacements` field is shown in Listing 520; as with all of the other fields, you are encouraged to customise and change this as you see fit. The options in this field will *only* be considered if the `-r`, `-rv` or `-rr` switches are active; when discussing YAML settings related to the replacement-mode switches, we will use the style given in Listing 520.

LISTING 520: `replacements`

-r

```
replacements:
  - amalgamate: 1
  - this: latexindent.pl
    that: pl.latexindent
    lookForThis: 0
    when: before
```

The first entry within the `replacements` field is `amalgamate`, and is *optional*; by default it is set to 1, so that replacements will be amalgamated from each settings file that you specify. As you'll see in the demonstrations that follow, there is no need to specify this field.

You'll notice that, by default, there is only *one* entry in the `replacements` field, but it can take as many entries as you would like; each one needs to begin with a `-` on its own line.

7.1 Introduction to replacements

Let's explore the action of the default settings, and then we'll demonstrate the feature with further examples.

example 147 Beginning with the code in Listing 521 and running the command

```
cmh:~$ latexindent.pl -r replace1.tex
```

gives the output given in Listing 522.

■

Listing 521: `replace1.tex`

```
Before text, latexindent.pl,
after text.
```

Listing 522: `replace1.tex` default

```
Before text, latexindent.pl,
after text.
```

We note that in Listing 520, because `lookForThis` is set to 0, the specified replacement has *not* been made, and there is no difference between Listings 521 and 522.

If we *do* wish to perform this replacement, then we can tweak the default settings of Listing 520 on the previous page by changing `lookForThis` to 1; we perform this action in Listing 524, and run the command

```
cmh:~$ latexindent.pl -r replace1.tex -l=replace1.yaml
```

which gives the output in Listing 523.

Listing 523: `replace1.tex` using Listing 524

```
Before text, pl.latexindent,
after text.
```

Listing 524: `replace1.yaml` -r

```
replacements:
  -
    amalgamate: 0
  -
    this: latexindent.pl
    that: pl.latexindent
    lookForThis: 1
```

Note that in Listing 524 we have specified `amalgamate` as 0 so that the default replacements are overwritten.

We haven't yet discussed the `when` field; don't worry, we'll get to it as part of the discussion in what follows.

7.2 The two types of replacements

There are two types of replacements:

1. *string*-based replacements, which replace the string in *this* with the string in *that*. If you specify `this` and you do not specify `that`, then the `that` field will be assumed to be empty.
2. *regex*-based replacements, which use the `substitution` field.

We will demonstrate both in the examples that follow.

`latexindent.pl` chooses which type of replacement to make based on which fields have been specified; if the `this` field is specified, then it will make *string*-based replacements, regardless of if `substitution` is present or not.

7.3 Examples of replacements

example 148 We begin with code given in Listing 525

Listing 525: `colsep.tex`

```
\begin{env}
1 2 3\arraycolsep=3pt
4 5 6\arraycolsep=5pt
\end{env}
```

Let's assume that our goal is to remove both of the `arraycolsep` statements; we can achieve this in a few different ways.

Using the YAML in Listing 527, and running the command

```
cmh:~$ latexindent.pl -r colsep.tex -l=colsep.yaml
```

then we achieve the output in Listing 526.

LISTING 526: `colsep.tex` using Listing 527

```
\begin{env}
   1 2 3
   4 5 6
\end{env}
```

LISTING 527: `colsep.yaml` -r

```
replacements:
  -
    this: \arraycolsep=3pt
  -
    this: \arraycolsep=5pt
```

Note that in Listing 527, we have specified *two* separate fields, each with their own '*this*' field; furthermore, for both of the separate fields, we have not specified '`that`', so the `that` field is assumed to be blank by `latexindent.pl`;

We can make the YAML in Listing 527 more concise by exploring the `substitution` field. Using the settings in Listing 529 and running the command

```
cmh:~$ latexindent.pl -r colsep.tex -l=colsep1.yaml
```

then we achieve the output in Listing 528.

LISTING 528: `colsep.tex` using Listing 529

```
\begin{env}
   1 2 3
   4 5 6
\end{env}
```

LISTING 529: `colsep1.yaml` -r

```
replacements:
  -
    substitution:
    s/\\arraycolsep=\d+pt//sg
```

The code given in Listing 529 is an example of a *regular expression*, which we may abbreviate to *regex* in what follows. This manual is not intended to be a tutorial on regular expressions; you might like to read, for example, [35] for a detailed covering of the topic. With reference to Listing 529, we do note the following:

- the general form of the `substitution` field is `s/regex/replacement/modifiers`. You can place any regular expression you like within this;
- we have 'escaped' the backslash by using `\\`
- we have used `\d+` to represent *at least* one digit
- the `s` *modifier* (in the `sg` at the end of the line) instructs `latexindent.pl` to treat your file as one single line;
- the `g` *modifier* (in the `sg` at the end of the line) instructs `latexindent.pl` to make the substitution *globally* throughout your file; you might try removing the `g` modifier from Listing 529 and observing the difference in output.

You might like to see https://perldoc.perl.org/perlre.html#Modifiers for details of modifiers; in general, I recommend starting with the `sg` modifiers for this feature. ■

example 149 We'll keep working with the file in Listing 525 on the preceding page for this example.

Using the YAML in Listing 531, and running the command

```
cmh:~$ latexindent.pl -r colsep.tex -l=multi-line.yaml
```

then we achieve the output in Listing 530. ■

LISTING 530: `colsep.tex` using Listing 531

```
multi-line!
```

LISTING 531: `multi-line.yaml` -r

```
replacements:
  -
    this: |-
     \begin{env}
     1 2 3\arraycolsep=3pt
     4 5 6\arraycolsep=5pt
     \end{env}
    that: ’multi-line!’
```

With reference to Listing 531, we have specified a *multi-line* version of `this` by employing the *literal* YAML style `|-`. See, for example, https://stackoverflow.com/questions/3790454/in-yaml-how-do-i-break-a-string-over-multiple-lines for further options, all of which can be used in your YAML file.

This is a natural point to explore the `when` field, specified in Listing 520 on page 129. This field can take two values: *before* and *after*, which respectively instruct `latexindent.pl` to perform the replacements *before* indentation or *after* it. The default value is `before`.

Using the YAML in Listing 533, and running the command

```
cmh:~$ latexindent.pl -r colsep.tex -l=multi-line1.yaml
```

then we achieve the output in Listing 532.

LISTING 532: `colsep.tex` using Listing 533

```
\begin{env}
   1 2 3\arraycolsep=3pt
   4 5 6\arraycolsep=5pt
\end{env}
```

LISTING 533: `multi-line1.yaml` -r

```
replacements:
  -
    this: |-
     \begin{env}
     1 2 3\arraycolsep=3pt
     4 5 6\arraycolsep=5pt
     \end{env}
    that: ’multi-line!’
    when: after
```

We note that, because we have specified `when: after`, that `latexindent.pl` has not found the string specified in Listing 533 within the file in Listing 525 on page 130. As it has looked for the string within Listing 533 *after* the indentation has been performed. After indentation, the string as written in Listing 533 is no longer part of the file, and has therefore not been replaced.

As a final note on this example, if you use the `-rr` switch, as follows,

```
cmh:~$ latexindent.pl -rr colsep.tex -l=multi-line1.yaml
```

then the `when` field is ignored, no indentation is done, and the output is as in Listing 530.

example 150 An important part of the substitution routine is in *capture groups*.

Assuming that we start with the code in Listing 534, let’s assume that our goal is to replace each occurrence of `$$...$$` with `\begin{equation*}...\end{equation*}`. This example is partly motivated by tex stackexchange question 242150.

LISTING 534: `displaymath.tex`

```
before text $$a^2+b^2=4$$ and $$c^2$$

$$
d^2+e^2 = f^2
$$
and also $$ g^2
$$ and some inline math: $h^2$
```

We use the settings in Listing 536 and run the command

```
cmh:~$ latexindent.pl -r displaymath.tex -l=displaymath1.yaml
```

to receive the output given in Listing 535.

LISTING 535: `displaymath.tex` using Listing 536

```
before text \begin{equation*}a^2+b^2=4\end{equation*}
    and \begin{equation*}c^2\end{equation*}

\begin{equation*}
   d^2+e^2 = f^2
\end{equation*}
and also \begin{equation*} g^2
\end{equation*} and some inline math: $h^2$
```

LISTING 536: `displaymath1.yaml`

-r

```
replacements:
  -
    substitution: |-
     s/\$\$
       (.*?)
       \$\$/\\begin{equation*}$1\\end{equation*}/sgx
```

A few notes about Listing 536:

1. we have used the `x` modifier, which allows us to have white space within the regex;
2. we have used a capture group, `(.*?)` which captures the content between the `$$...$$` into the special variable, `$1`;
3. we have used the content of the capture group, `$1`, in the replacement text.

See https://perldoc.perl.org/perlre.html#Capture-groups for a discussion of capture groups.

The features of the replacement switches can, of course, be combined with others from the toolkit of `latexindent.pl`. For example, we can combine the poly-switches of Section 6.3 on page 108, which we do in Listing 538; upon running the command

```
cmh:~$ latexindent.pl -r -m displaymath.tex -l=displaymath1.yaml,equation.yaml
```

then we receive the output in Listing 537.

LISTING 537: displaymath.tex using Listings 536 and 538

```
before text%
\begin{equation*}%
   a^2+b^2=4%
\end{equation*}%
and%
\begin{equation*}%
   c^2%
\end{equation*}

\begin{equation*}
   d^2+e^2 = f^2
\end{equation*}
and also%
\begin{equation*}%
   g^2
\end{equation*}%
and some inline math: $h^2$
```

LISTING 538: equation.yaml -m

```
modifyLineBreaks:
    environments:
        equation*:
            BeginStartsOnOwnLine: 2
            BodyStartsOnOwnLine: 2
            EndStartsOnOwnLine: 2
            EndFinishesWithLineBreak: 2
```

example 151 This example is motivated by tex stackexchange question 490086. We begin with the code in Listing 539.

LISTING 539: phrase.tex

```
phrase 1       phrase 2 phrase 3             phrase 100

phrase 1       phrase 2 phrase 3             phrase 100

phrase 1       phrase 2 phrase 3             phrase 100

phrase 1       phrase 2 phrase 3             phrase 100
```

Our goal is to make the spacing uniform between the phrases. To achieve this, we employ the settings in Listing 541, and run the command

```
cmh:~$ latexindent.pl -r phrase.tex -l=hspace.yaml
```

which gives the output in Listing 540.

LISTING 540: phrase.tex using Listing 541

```
phrase 1 phrase 2 phrase 3 phrase 100

phrase 1 phrase 2 phrase 3 phrase 100

phrase 1 phrase 2 phrase 3 phrase 100

phrase 1 phrase 2 phrase 3 phrase 100
```

LISTING 541: hspace.yaml -r

```
replacements:
  -
    substitution: s/\h+/ /sg
```

The `\h+` setting in Listing 541 say to replace *at least one horizontal space* with a single space. ■

example 152 We begin with the code in Listing 542. ■

LISTING 542: references.tex

```
equation \eqref{eq:aa} and Figure  \ref{fig:bb}
and table~\ref{tab:cc}
```

Our goal is to change each reference so that both the text and the reference are contained within one hyperlink. We achieve this by employing Listing 544 and running the command

```
cmh:~$ latexindent.pl -r references.tex -l=reference.yaml
```

which gives the output in Listing 543.

LISTING 543: references.tex using Listing 544

```
\hyperref{equation \ref*{eq:aa}} and \hyperref{Figure \ref*{fig:bb}}
and \hyperref{table \ref*{tab:cc}}
```

LISTING 544: reference.yaml -r

```
replacements:
  -
    substitution: |-
      s/(
        equation
        |
        table
        |
        figure
        |
        section
      )
      (\h|~)*
      \\(?:eq)?
      ref\{(.*?)\}/\\hyperref{$1 \\ref\*{$3}}/sgxi
```

Referencing Listing 544, the `|` means *or*, we have used *capture groups*, together with an example of an *optional* pattern, `(?:eq)?`.

example 153 Let’s explore the three replacement mode switches (see Table 4 on page 129) in the context of an example that contains a verbatim code block, Listing 545; we will use the settings in Listing 546.

LISTING 545: verb1.tex

```
\begin{myenv}
body of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    body
        of
      verbatim
 text
\end{verbatim}
text
```

LISTING 546: verbatim1.yaml -r

```
replacements:
  -
    this: ’body’
    that: ’head’
```

Upon running the following commands,

```
cmh:~$ latexindent.pl -r verb1.tex -l=verbatim1.yaml -o=+mod1
cmh:~$ latexindent.pl -rv verb1.tex -l=verbatim1.yaml -o=+-rv-mod1
cmh:~$ latexindent.pl -rr verb1.tex -l=verbatim1.yaml -o=+-rr-mod1
```

we receive the respective output in Listings 547 to 549

LISTING 547: `verb1-mod1.tex`

```
\begin{myenv}
   head of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    head
        of
      verbatim
 text
\end{verbatim}
text
```

LISTING 548: `verb1-rv-mod1.tex`

```
\begin{myenv}
   head of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    body
        of
      verbatim
 text
\end{verbatim}
text
```

LISTING 549: `verb1-rr-mod1.tex`

```
\begin{myenv}
head of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    head
        of
      verbatim
 text
\end{verbatim}
text
```

We note that:

1. in Listing 547 indentation has been performed, and that the replacements specified in Listing 546 have been performed, even within the verbatim code block;
2. in Listing 548 indentation has been performed, but that the replacements have *not* been performed within the verbatim environment, because the `rv` switch is active;
3. in Listing 549 indentation has *not* been performed, but that replacements have been performed, not respecting the verbatim code block.

See the summary within Table 4 on page 129.

example 154 Let's explore the `amalgamate` field from Listing 520 on page 129 in the context of the file specified in Listing 550.

LISTING 550: `amalg1.tex`

```
one two three
```

Let's consider the YAML files given in Listings 551 to 553.

LISTING 551: `amalg1-yaml.yaml` -r

```
replacements:
  -
    this: one
    that: 1
```

LISTING 552: `amalg2-yaml.yaml` -r

```
replacements:
  -
    this: two
    that: 2
```

LISTING 553: `amalg3-yaml.yaml` -r

```
replacements:
  -
    amalgamate: 0
  -
    this: three
    that: 3
```

Upon running the following commands,

```
cmh:~$ latexindent.pl -r amalg1.tex -l=amalg1-yaml
cmh:~$ latexindent.pl -r amalg1.tex -l=amalg1-yaml,amalg2-yaml
cmh:~$ latexindent.pl -r amalg1.tex -l=amalg1-yaml,amalg2-yaml,amalg3-yaml
```

we receive the respective output in Listings 554 to 556.

LISTING 554: `amalg1.tex` using Listing 551

```
1 two three
```

LISTING 555: `amalg1.tex` using Listings 551 and 552

```
1 2 three
```

LISTING 556: `amalg1.tex` using Listings 551 to 553

```
one two 3
```

We note that:

1. in Listing 554 the replacements from Listing 551 have been used;
2. in Listing 555 the replacements from Listings 551 and 552 have *both* been used, because

the default value of `amalgamate` is 1;

3. in Listing 556 *only* the replacements from Listing 553 have been used, because the value of `amalgamate` has been set to 0.

■

SECTION 8

The –lines switch

N: 2021-09-16

`latexindent.pl` can operate on a *selection* of lines of the file using the `-lines` or `-n` switch.

The basic syntax is `-lines MIN-MAX`, so for example

```
cmh:~$ latexindent.pl --lines 3-7 myfile.tex
cmh:~$ latexindent.pl -n 3-7 myfile.tex
```

will only operate upon lines 3 to 7 in `myfile.tex`. All of the other lines will *not* be operated upon by `latexindent.pl`.

The options for the `lines` switch are:

- line range, as in `-lines 3-7`
- single line, as in `-lines 5`
- multiple line ranges separated by commas, as in `-lines 3-5,8-10`
- negated line ranges, as in `-lines !3-5` which translates to `-lines 1-2,6-N`, where N is the number of lines in your file.

We demonstrate this feature, and the available variations in what follows. We will use the file in Listing 557.

LISTING 557: `myfile.tex`

```
1  Before the environments
2  \begin{one}
3     first block, first line
4     first block, second line
5     first block, third line
6     \begin{two}
7        second block, first line
8        second block, second line
9        second block, third line
10       second block, fourth line
11    \end{two}
12 \end{one}
```

example 155 We demonstrate the basic usage using the command

```
cmh:~$ latexindent.pl --lines 3-7 myfile.tex -o=+-mod1
```

which instructs `latexindent.pl` to only operate on lines 3 to 7; the output is given in Listing 558. ■

Listing 558: `myfile-mod1.tex`

```
1  Before the environments
2  \begin{one}
3  first block, first line
4  first block, second line
5  first block, third line
6  \begin{two}
7  second block, first line
8        second block, second line
9        second block, third line
10       second block, fourth line
11    \end{two}
12 \end{one}
```

The following two calls to `latexindent.pl` are equivalent

```
cmh:~$ latexindent.pl --lines 3-7 myfile.tex -o=+-mod1
cmh:~$ latexindent.pl --lines 7-3 myfile.tex -o=+-mod1
```

as `latexindent.pl` performs a check to put the lowest number first.

example 156 You can call the `lines` switch with only *one number* and in which case only that line will be operated upon. For example

```
cmh:~$ latexindent.pl --lines 5 myfile.tex -o=+-mod2
```

instructs `latexindent.pl` to only operate on line 5; the output is given in Listing 559.

Listing 559: `myfile-mod2.tex`

```
1  Before the environments
2  \begin{one}
3     first block, first line
4     first block, second line
5  first block, third line
6     \begin{two}
7        second block, first line
8        second block, second line
9        second block, third line
10       second block, fourth line
11    \end{two}
12 \end{one}
```

The following two calls are equivalent:

```
cmh:~$ latexindent.pl --lines 5 myfile.tex
cmh:~$ latexindent.pl --lines 5-5 myfile.tex
```

example 157 If you specify a value outside of the line range of the file then `latexindent.pl` will ignore the `lines` argument, detail as such in the log file, and proceed to operate on the entire file.

For example, in the following call

```
cmh:~$ latexindent.pl --lines 11-13 myfile.tex
```

`latexindent.pl` will ignore the `lines` argument, and *operate on the entire file* because Listing 557 only has 12 lines.

Similarly, in the call

```
cmh:~$ latexindent.pl --lines -1-3 myfile.tex
```

`latexindent.pl` will ignore the `lines` argument, and *operate on the entire file* because we assume that negatively numbered lines in a file do not exist.

example 158 You can specify *multiple line ranges* as in the following

```
cmh:~$ latexindent.pl --lines 3-5,8-10 myfile.tex -o=+-mod3
```

which instructs `latexindent.pl` to operate upon lines 3 to 5 and lines 8 to 10; the output is given in Listing 560.

LISTING 560: `myfile-mod3.tex`

```
1  Before the environments
2  \begin{one}
3  first block, first line
4  first block, second line
5  first block, third line
6     \begin{two}
7        second block, first line
8  second block, second line
9  second block, third line
10 second block, fourth line
11    \end{two}
12 \end{one}
```

The following calls to `latexindent.pl` are all equivalent

```
cmh:~$ latexindent.pl --lines 3-5,8-10 myfile.tex
cmh:~$ latexindent.pl --lines 8-10,3-5 myfile.tex
cmh:~$ latexindent.pl --lines 10-8,3-5 myfile.tex
cmh:~$ latexindent.pl --lines 10-8,5-3 myfile.tex
```

as `latexindent.pl` performs a check to put the lowest line ranges first, and within each line range, it puts the lowest number first.

example 159 There's no limit to the number of line ranges that you can specify, they just need to be separated by commas. For example

```
cmh:~$ latexindent.pl --lines 1-2,4-5,9-10,12 myfile.tex -o=+-mod4
```

has four line ranges: lines 1 to 2, lines 4 to 5, lines 9 to 10 and line 12. The output is given in Listing 561.

Listing 561: `myfile-mod4.tex`

```
Before the environments
\begin{one}
   first block, first line
   first block, second line
   first block, third line
   \begin{two}
      second block, first line
      second block, second line
   second block, third line
   second block, fourth line
   \end{two}
\end{one}
```

As previously, the ordering does not matter, and the following calls to `latexindent.pl` are all equivalent

```
cmh:~$ latexindent.pl --lines 1-2,4-5,9-10,12 myfile.tex
cmh:~$ latexindent.pl --lines 2-1,4-5,9-10,12 myfile.tex
cmh:~$ latexindent.pl --lines 4-5,1-2,9-10,12 myfile.tex
cmh:~$ latexindent.pl --lines 12,4-5,1-2,9-10 myfile.tex
```

as `latexindent.pl` performs a check to put the lowest line ranges first, and within each line range, it puts the lowest number first.

example 160 You can specify *negated line ranges* by using `!` as in

```
cmh:~$ latexindent.pl --lines !5-7 myfile.tex -o=+-mod5
```

which instructs `latexindent.pl` to operate upon all of the lines *except* lines 5 to 7.

In other words, `latexindent.pl` *will* operate on lines 1 to 4, and 8 to 12, so the following two calls are equivalent:

```
cmh:~$ latexindent.pl --lines !5-7 myfile.tex
cmh:~$ latexindent.pl --lines 1-4,8-12 myfile.tex
```

The output is given in Listing 562.

Listing 562: `myfile-mod5.tex`

```
Before the environments
\begin{one}
   first block, first line
   first block, second line
   first block, third line
   \begin{two}
      second block, first line
   second block, second line
   second block, third line
   second block, fourth line
   \end{two}
\end{one}
```

example 161 You can specify *multiple negated line ranges* such as

```
cmh:~$ latexindent.pl --lines !5-7,!9-10 myfile.tex -o=+-mod6
```

which is equivalent to:

```
cmh:~$ latexindent.pl --lines 1-4,8,11-12 myfile.tex -o=+-mod6
```

The output is given in Listing 563.

LISTING 563: `myfile-mod6.tex`

```
Before the environments
\begin{one}
   first block, first line
   first block, second line
   first block, third line
   \begin{two}
      second block, first line
   second block, second line
      second block, third line
      second block, fourth line
   \end{two}
\end{one}
```

■

example 162 If you specify a line range with anything other than an integer, then `latexindent.pl` will ignore the `lines` argument, and *operate on the entire file*.

Sample calls that result in the `lines` argument being ignored include the following:

```
cmh:~$ latexindent.pl --lines 1-x myfile.tex
cmh:~$ latexindent.pl --lines !y-3 myfile.tex
```

■

example 163 We can, of course, use the `lines` switch in combination with other switches.

For example, let's use with the file in Listing 564.

LISTING 564: `myfile1.tex`

```
Before the environments
\begin{one}
   first block, first line
   first block, second line
   first block, third line
   \begin{two} body \end{two}
\end{one}
```

We can demonstrate interaction with the `-m` switch (see Section 6 on page 79); in particular, if we use Listing 456 on page 114, Listing 440 on page 112 and Listing 441 on page 112 and run

```
cmh:~$ latexindent.pl --lines 6 myfile1.tex -o=+-mod1 -m -l env-mlb2,env-mlb7,env-mlb8 -o=+-mod1
```

then we receive the output in Listing 565.

■

LISTING 565: `myfile1-mod1.tex`

```
Before the environments
\begin{one}
   first block, first line
   first block, second line
   first block, third line
\begin{two}
   body
\end{two}
\end{one}
```

■

SECTION 9

Fine tuning

N: 2019-07-13

`latexindent.pl` operates by looking for the code blocks detailed in Table 2 on page 56. The fine tuning of the details of such code blocks is controlled by the `fineTuning` field, detailed in Listing 566.

This field is for those that would like to peek under the bonnet/hood and make some fine tuning to `latexindent.pl`'s operating.

Warning!

Making changes to the fine tuning may have significant consequences for your indentation scheme, proceed with caution!

LISTING 566: `fineTuning`

```
fineTuning:
  environments:
    name: [a-zA-Z@\*0-9_\\]+
  ifElseFi:
    name: (?!@?if[a-zA-Z@]*?\{)@?if[a-zA-Z@]*?
  commands:
    name: [+a-zA-Z@\*0-9_\:]+?
  items:
    canBeFollowedBy: (?:\[[^]]*?\])|(?:<[^>]*?>)
  keyEqualsValuesBracesBrackets:
    name: [a-zA-Z@\*0-9_\/.:\#-]+[a-zA-Z@\*0-9_\/.\h\{\}:\#-]*?
    follow: (?:(?<!\\)\{)|,|(?:(?<!\\)\[)
  namedGroupingBracesBrackets:
    name: [0-9\.a-zA-Z@\*><]+?
    follow: \h|\R|\{|\[|\$|\)|\(
  UnNamedGroupingBracesBrackets:
    follow: \{|\[|,|&|\)|\(|\$
  arguments:
    before: (?:#\d\h*;?,?\/?)+|\<.*?\>
    between: _|\^|\*
  trailingComments:
    notPrecededBy: (?<!\\)
    afterComment: .*?
  modifyLineBreaks:
    doubleBackSlash: \\\\(?:\h*\[\h*\d+\h*[a-zA-Z]+\h*\])?
    comma: ','
    betterFullStop: |-
      (?x)                                # ignore spaces in the below
      (?:                                 #
        \.\)                              # .)
        (?!\h*[a-z])                      # not *followed by* a-z
      )                                   #
      |                                   # OR
      (?:                                 #
        (?<!                              # not *preceded by*
          (?:                             #
            (?:[eE]\.[gG])                # e.g OR E.g OR e.G OR E.G
            |                             #
            (?:[iI]\.[eE])                # i.e OR I.e OR i.E OR I.E
            |                             #
            (?:etc)                       # etc
            |                             #
            (?:[wW]\.[rR]\.[tT])          # w.r.t OR W.r.t OR w.R.t OR w.r.T OR W.R.t OR W.r.T OR w.R.T OR W.R.T
          )                               #
        )                                 #
      )                                   #
      \.                                  # .
      (?!                                 # not *followed by*
        (?:                               #
          [a-zA-Z0-9-~,]                  #
          |                               #
          \),                             # ),
          |                               #
          \)\.                            # ).
        )                                 #
      )                                   #
```

The fields given in Listing 566 are all *regular expressions*. This manual is not intended to be a tutorial on regular expressions; you might like to read, for example, [35] for a detailed covering of the topic.

We make the following comments with reference to Listing 566:

1. the `environments:name` field details that the *name* of an environment can contain:
    (a) `a-z` lower case letters
    (b) `A-Z` upper case letters
    (c) `@` the `@` 'letter'
    (d) `\*` stars
    (e) `0-9` numbers
    (f) `_` underscores
    (g) `\` backslashes

    The `+` at the end means *at least one* of the above characters.
2. the `ifElseFi:name` field:
    (a) `@?` means that it *can possibly* begin with `@`
    (b) followed by `if`
    (c) followed by 0 or more characters from `a-z`, `A-Z` and `@`
    (d) the `?` the end means *non-greedy*, which means 'stop the match as soon as possible'
3. the `keyEqualsValuesBracesBrackets` contains some interesting syntax:
    (a) `|` means 'or'
    (b) `(?:(?<!\\)\{)` the `(?:...)` uses a *non-capturing* group – you don't necessarily need to worry about what this means, but just know that for the `fineTuning` feature you should only ever use *non*-capturing groups, and *not* capturing groups, which are simply `(...)`
    (c) `(?<!\\)\{)` means a `{` but it can *not* be immediately preceded by a `\`
4. in the `arguments:before` field
    (a) `\d\h*` means a digit (i.e. a number), followed by 0 or more horizontal spaces
    (b) `;?,?` means *possibly* a semi-colon, and possibly a comma
    (c) `\<.*?\>` is designed for 'beamer'-type commands; the `.*?` means anything in between `<...>`

5. the `modifyLineBreaks` field refers to fine tuning settings detailed in Section 6 on page 79. In particular:
    (a) `betterFullStop` is in relation to the one sentence per line routine, detailed in Section 6.2 on page 95
    (b) `doubleBackSlash` is in relation to the `DBSStartsOnOwnLine` and `DBSFinishesWithLineBreak` polyswitches surrounding double backslashes, see Section 6.3.2 on page 118
    (c) `comma` is in relation to the `CommaStartsOnOwnLine` and `CommaFinishesWithLineBreak` polyswitches surrounding commas in optional and mandatory arguments; see Table 3 on page 123

It is not obvious from Listing 566, but each of the `follow`, `before` and `between` fields allow trailing comments, line breaks, and horizontal spaces between each character.

Warning!

For the `fineTuning` feature you should only ever use *non*-capturing groups, such as `(?:...)` and *not* capturing groups, which are `(...)`

example 164 As a demonstration, consider the file given in Listing 567, together with its default output using the command

```
cmh:~$ latexindent.pl finetuning1.tex
```

is given in Listing 568.

LISTING 567: `finetuning1.tex`

```
\mycommand{
    \rule{G -> +H[-G]CL}
    \rule{H -> -G[+H]CL}
    \rule{g -> +h[-g]cL}
    \rule{h -> -g[+h]cL}
}
```

LISTING 568: `finetuning1.tex` default

```
\mycommand{
\rule{G -> +H[-G]CL}
\rule{H -> -G[+H]CL}
\rule{g -> +h[-g]cL}
\rule{h -> -g[+h]cL}
}
```

It's clear from Listing 568 that the indentation scheme has not worked as expected. We can *fine tune* the indentation scheme by employing the settings given in Listing 570 and running the command

```
cmh:~$ latexindent.pl finetuning1.tex -l=fine-tuning1.yaml
```

and the associated (desired) output is given in Listing 569.

LISTING 569: `finetuning1.tex` using Listing 570

```
\mycommand{
   \rule{G -> +H[-G]CL}
   \rule{H -> -G[+H]CL}
   \rule{g -> +h[-g]cL}
   \rule{h -> -g[+h]cL}
}
```

LISTING 570: `finetuning1.yaml`

```
fineTuning:
    arguments:
      between:
    '_|\^|\*|\->|\-|\+|h|H|g|G'
```

■

example 165 Let's have another demonstration; consider the file given in Listing 571, together with its default output using the command

```
cmh:~$ latexindent.pl finetuning2.tex
```

■

is given in Listing 572.

LISTING 571: `finetuning2.tex`

```
@misc{ wikilatex,
author = "{Wikipedia contributors}",
title = "LaTeX --- {Wikipedia}{,}",
note = "[Online; accessed 3-March-2020]"
}
```

LISTING 572: `finetuning2.tex` default

```
@misc{ wikilatex,
author = "{Wikipedia contributors}",
title = "LaTeX --- {Wikipedia}{,}",
note = "[Online; accessed 3-March-2020]"
}
```

It's clear from Listing 572 that the indentation scheme has not worked as expected. We can *fine tune* the indentation scheme by employing the settings given in Listing 574 and running the command

```
cmh:~$ latexindent.pl finetuning2.tex -l=fine-tuning2.yaml
```

and the associated (desired) output is given in Listing 573.

LISTING 573: `finetuning2.tex` using Listing 574

```
@misc{ wikilatex,
   author = "{Wikipedia contributors}",
   title = "LaTeX --- {Wikipedia}{,}",
   note = "[Online; accessed 3-March-2020]"
}
```

LISTING 574: `finetuning2.yaml`

```
fineTuning:
    NamedGroupingBracesBrackets:
      follow: '\h|\R|\{|\[|\$|\)|\(|"'
    UnNamedGroupingBracesBrackets:
      follow: '\{|\[|,|&|\)|\(|\$|"'
    arguments:
      between: '_|\^|\*|---'
```

In particular, note that the settings in Listing 574 specify that `NamedGroupingBracesBrackets` and `UnNamedGroupingBracesBrackets` can follow " and that we allow --- between arguments.

example 166 You can tweak the `fineTuning` using the -y switch, but to be sure to use quotes appropriately. For example, starting with the code in Listing 575 and running the following command

```
cmh:~$ latexindent.pl -m
    -y='modifyLineBreaks:oneSentencePerLine:manipulateSentences:␣1,␣
    modifyLineBreaks:oneSentencePerLine:sentencesBeginWith:a-z:␣1,␣
    fineTuning:modifyLineBreaks:betterFullStop:␣
    "(?:\.|;|:(?![a-z]))|(?:(?<!(?:(?:e\.g)|(?:i\.e)|(?:etc)))\.(?!(?:[a-z]|[A-Z]|\
    issue-243.tex -o=+-mod1
```

gives the output shown in Listing 576.

LISTING 575: `finetuning3.tex`

```
We go; you see: this sentence \cite{tex:stackexchange} finishes here.
```

LISTING 576: `finetuning3.tex` using -y switch

```
We go;
you see:
this sentence \cite{tex:stackexchange} finishes here.
```

example 167 We can tweak the `fineTuning` for how trailing comments are classified. For motivation, let's consider the code given in Listing 577

LISTING 577: `finetuning4.tex`

```
some before text
 \href{Handbook%20for%30Spoken%40document.pdf}{my document}
some after text
```

We will compare the settings given in Listings 578 and 579.

LISTING 578: `href1.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: -1
        blocksEndBefore:
           verbatim: 0
        blocksFollow:
           verbatim: 0

removeTrailingWhitespace:
    beforeProcessing: 1
```

LISTING 579: `href2.yaml` -m

```
fineTuning:
    trailingComments:
      notPrecededBy:
    '(?:(?<!Handbook)(?<!for)(?<!Spoken))'

modifyLineBreaks:
    textWrapOptions:
        columns: -1
        blocksEndBefore:
           verbatim: 0
        blocksFollow:
           verbatim: 0

removeTrailingWhitespace:
    beforeProcessing: 1
```

Upon running the following commands

```
cmh:~$ latexindent.pl -m finetuning4.tex -o=+-mod1 -l=href1
cmh:~$ latexindent.pl -m finetuning4.tex -o=+-mod2 -l=href2
```

we receive the respective output in Listings 580 and 581.

LISTING 580: `finetuning4.tex` using Listing 578

```
some before text \href{Handbooksome after text%20for%30Spoken%40document.pdf}{my document}
```

LISTING 581: `finetuning4.tex` using Listing 579

```
some before text \href{Handbook%20for%30Spoken%40document.pdf}{my document} some after text
```

We note that in:

- Listing 580 the trailing comments are assumed to be everything following the first comment symbol, which has meant that everything following it has been moved to the end of the line; this is undesirable, clearly!
- Listing 581 has fine-tuned the trailing comment matching, and says that % cannot be immediately preceded by the words ‘Handbook’, ‘for’ or ‘Spoken’, which means that none of the % symbols have been treated as trailing comments, and the output is desirable. ■

example 168 Another approach to this situation, which does not use `fineTuning`, is to use `noIndentBlock` which we discussed in Listing 43 on page 30; using the settings in Listing 582 and running the command

```
cmh:~$ latexindent.pl -m finetuning4.tex -o=+-mod3 -l=href3
```

then we receive the same output given in Listing 581. ■

Listing 582: `href3.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: -1
        blocksEndBefore:
           verbatim: 0
        blocksFollow:
           verbatim: 0

noIndentBlock:
    href:
        begin: '\\href\{[^}]*?\}\{'
        body: '[^}]*?'
        end: '\}'
```

With reference to the `body` field in Listing 582, we note that the `body` field can be interpreted as: the fewest number of zero or more characters that are not right braces. This is an example of character class.

example 169 We can use the `fineTuning` field to assist in the formatting of bibliography files.

Starting with the file in Listing 583 and running the command

```
cmh:~$ latexindent.pl bib1.tex -o=+-mod1
```

gives the output in Listing 584.

Listing 583: `bib1.bib`

```
@online{paulo,
title="arararule,indent.yaml",
author="PauloCereda",
date={2013-05-23},
urldate={2021-03-19},
keywords={contributor},}
```

Listing 584: `bib1-mod1.bib`

```
@online{paulo,
   title="arararule,indent.yaml",
   author="PauloCereda",
   date={2013-05-23},
   urldate={2021-03-19},
   keywords={contributor},}
```

Let's assume that we would like to format the output so as to align the = symbols. Using the settings in Listing 586 and running the command

```
cmh:~$ latexindent.pl bib1.bib -l bibsettings1.yaml -o=+-mod2
```

gives the output in Listing 585.

Listing 585: `bib1.bib` using Listing 586

```
@online{paulo,
   title    = "arararule,indent.yaml",
   author   = "PauloCereda",
   date     = {2013-05-23},
   urldate  = {2021-03-19},
   keywords = {contributor},}
```

Listing 586: `bibsettings1.yaml`

```
lookForAlignDelims:
   online:
      delimiterRegEx: '(=)'

fineTuning:
    keyEqualsValuesBracesBrackets:
      follow:
    '(?:(?<!\\)\{)|(?:(?<!\\)\[)'
    UnNamedGroupingBracesBrackets:
      follow: '\{|\[|,|&|\)|\(|\$|='
```

Some notes about Listing 586:

- we have populated the `lookForAlignDelims` field with the `online` command, and have used the `delimiterRegEx`, discussed in Section 5.5.4 on page 43;
- we have tweaked the `keyEqualsValuesBracesBrackets` code block so that it will *not* be found following a comma; this means that, in contrast to the default behaviour, the lines such as `date={2013-05-23},` will *not* be treated as key-equals-value braces;
- the adjustment to `keyEqualsValuesBracesBrackets` necessitates the associated change to the `UnNamedGroupingBracesBrackets` field so that they will be searched for following = symbols.

example 170 We can build upon Listing 586 for slightly more complicated bibliography files.

Starting with the file in Listing 587 and running the command

```
cmh:~$ latexindent.pl bib2.bib -l bibsettings1.yaml -o=+-mod1
```

gives the output in Listing 588.

Listing 587: `bib2.bib`

```
@online{cmh:videodemo,
title="Videodemonstrationofpl.latexindentonyoutube",
url="https://www.youtube.com/watch?v=wo38aaH2F4E&spfreload=10",
urldate={2017-02-21},
}
```

Listing 588: `bib2-mod1.bib`

```
@online{cmh:videodemo,
   title   = "Videodemonstrationofpl.latexindentonyoutube",
   url     = "https://www.youtube.com/watch?v               = wo38aaH2F4E&spfreload = 10",
   urldate = {2017-02-21},
}
```

The output in Listing 588 is not ideal, as the = symbol within the url field has been incorrectly used as an alignment delimiter.

We address this by tweaking the `delimiterRegEx` field in Listing 589.

Listing 589: `bibsettings2.yaml`

```
lookForAlignDelims:
   online:
      delimiterRegEx: '(?<!v)(?<!spfreload)(=)'
```

Upon running the command

```
cmh:~$ latexindent.pl bib2.bib -l bibsettings1.yaml,bibsettings2.yaml -o=+-mod2
```

we receive the *desired* output in Listing 590.

Listing 590: `bib2-mod2.bib`

```
@online{cmh:videodemo,
   title   = "Videodemonstrationofpl.latexindentonyoutube",
   url     = "https://www.youtube.com/watch?v=wo38aaH2F4E&spfreload=10",
   urldate = {2017-02-21},
}
```

With reference to Listing 589 we note that the `delimiterRegEx` has been adjusted so that = symbols are used as the delimiter, but only when they are *not preceded* by either `v` or `spfreload`.

N: 2023-06-01 **ample 171** We can use the `fineTuning` settings to tweak how `latexindent.pl` finds trailing comments.

We begin with the file in Listing 591

LISTING 591: `finetuning5.tex`

```
\chapter{chapter text} % 123
chapter text
\section{section text} % 456
section text
% end
% end
```

Using the settings in Listing 593 and running the command

```
cmh:~$ latexindent.pl finetuning5.tex -l=fine-tuning3.yaml
```

gives the output in Listing 592.

LISTING 592: `finetuning5-mod1.tex`

```
\chapter{chapter text} % 123
   chapter text
   \section{section text} % 456
      section text
   % end
% end
```

LISTING 593: `finetuning3.yaml`

```
fineTuning:
    trailingComments:
      notPrecededBy: (?<!\\)
      afterComment: (?!(?:\hend)).*?

specialBeginEnd:
  customSection:
    begin: \\(?:section|chapter)
    end: \%\h+end
  specialBeforeCommand: 1
```

The settings in Listing 593 detail that trailing comments can *not* be followed by a single space, and then the text 'end'. This means that the `specialBeginEnd` routine will be able to find the pattern `% end` as the end part. The trailing comments `123` and `456` are still treated as trailing comments.

■

example 172 We can use the `fineTuning` settings to tweak how `latexindent.pl` finds environments.

N: 2023-10-13

We begin with the file in Listing 594.

LISTING 594: `finetuning6.tex`

```
\begin{myenv}\label{mylabel}The body of my environment...\end{myenv}
```

Using the settings in Listing 596 and running the command

```
cmh:~$ latexindent.pl finetuning6.tex -m -l=fine-tuning4.yaml
```

gives the output in Listing 595.

LISTING 595: `finetuning6-mod1.tex`

```
\begin{myenv}\label{mylabel}
   The body of my environment...
\end{myenv}
```

LISTING 596: `fine-tuning4.yaml`

-m

```
modifyLineBreaks:
   environments:
       BodyStartsOnOwnLine: 1
       EndStartsOnOwnLine: 1

fineTuning:
  environments:
    begin: \\begin\{([a-zA-Z@\*0-9_\\]+)\}\s*\\label\{[^}]+?\}
    end: \\end\{\2\}
```

■

By using the settings in Listing 596 it means that the default poly-switch location of `BodyStartsOnOwnLine` for environments (denoted ♡ in Table 3) has been overwritten so that it is *after* the `label` command.

Referencing Listing 596, unless both `begin` and `end` are specified, then the default value of `name` will be used. ■

SECTION 10

Conclusions and known limitations

There are a number of known limitations of the script, and almost certainly quite a few that are *unknown*! The known issues include:

multicolumn alignment when working with code blocks in which multicolumn commands overlap, the algorithm can fail; see Listing 71 on page 36.

textWrap after when operating with `indentRules` (see Section 5.8 on page 55) may not always cooperate with one another; if you have a specific example that does not work, please report it to [36].

efficiency particularly when the `-m` switch is active, as this adds many checks and processes. The current implementation relies upon finding and storing *every* code block (see the discussion on page 127); I hope that, in a future version, only *nested* code blocks will need to be stored in the 'packing' phase, and that this will improve the efficiency of the script.

U: 2019-07-13

You can run `latexindent` on any file; if you don't specify an extension, then the extensions that you specify in `fileExtensionPreference` (see Listing 35 on page 27) will be consulted. If you find a case in which the script struggles, please feel free to report it at [36], and in the meantime, consider using a `noIndentBlock` (see page 30).

I hope that this script is useful to some; if you find an example where the script does not behave as you think it should, the best way to contact me is to report an issue on [36]; otherwise, feel free to find me on the `http://tex.stackexchange.com/users/6621/cmhughes`.

SECTION 11

References

11.1 perl-related links

[32] *CPAN: Comprehensive Perl Archive Network*. URL: http://www.cpan.org/ (visited on 01/23/2017).

[33] *Data Dumper demonstration*. URL: https://stackoverflow.com/questions/7466825/how-do-you-sort-the-output-of-datadumper (visited on 06/18/2021).

[34] *Data::Dumper module*. URL: https://perldoc.perl.org/Data::Dumper (visited on 06/18/2021).

[35] Jeffrey E. F. Friedl. *Mastering Regular Expressions*. ISBN: 0596002890.

[41] *Log4perl Perl module*. URL: http://search.cpan.org/~mschilli/Log-Log4perl-1.49/lib/Log/Log4perl.pm (visited on 09/24/2017).

[42] *Perlbrew*. URL: http://perlbrew.pl/ (visited on 01/23/2017).

[43] *perldoc Encode::Supported*. URL: https://perldoc.perl.org/Encode::Supported (visited on 05/06/2021).

[46] *Strawberry Perl*. URL: http://strawberryperl.com/ (visited on 01/23/2017).

[47] *Text::Tabs Perl module*. URL: http://search.cpan.org/~muir/Text-Tabs+Wrap-2013.0523/lib.old/Text/Tabs.pm (visited on 07/06/2017).

[48] *Text::Wrap Perl module*. URL: http://perldoc.perl.org/Text/Wrap.html (visited on 05/01/2017).

11.2 conda-related links

[30] *anacoda*. URL: https://www.anaconda.com/products/individual (visited on 12/22/2021).

[31] *conda forge*. URL: https://github.com/conda-forge/miniforge (visited on 12/22/2021).

[38] *How to install Anaconda on Ubuntu?* URL: https://askubuntu.com/questions/505919/how-to-install-anaconda-on-ubuntu (visited on 01/21/2022).

[45] *Solving environment: failed with initial frozen solve. Retrying with flexible solve.* URL: https://github.com/conda/conda/issues/9367#issuecomment-558863143 (visited on 01/21/2022).

11.3 VScode-related links

[37] *How to create your own auto-completion for JSON and YAML files on VS Code with the help of JSON Schema*. URL: https://dev.to/brpaz/how-to-create-your-own-auto-completion-for-json-and-yaml-files-on-vs-code-with-the-help-of-json-schema-k1i (visited on 01/01/2022).

[50] *VSCode YAML extension*. URL: https://marketplace.visualstudio.com/items?itemName=redhat.vscode-yaml (visited on 01/01/2022).

11.4 Other links

[29] *A Perl script for indenting tex files*. URL: http://tex.blogoverflow.com/2012/08/a-perl-script-for-indenting-tex-files/ (visited on 01/23/2017).

[36] *Home of latexindent.pl*. URL: https://github.com/cmhughes/latexindent.pl (visited on 01/23/2017).

[39] *How to use latexindent on Windows?* URL: https://tex.stackexchange.com/questions/577250/how-to-use-latexindent-on-windows (visited on 01/08/2022).

[40] *latexindent.pl ghcr (GitHub Container Repository) location*. URL: https://github.com/cmhughes?tab=packages (visited on 06/12/2022).

[44] *pre-commit: A framework for managing and maintaining multi-language pre-commit hooks*. URL: https://pre-commit.com/ (visited on 01/08/2022).

[49] *Video demonstration of latexindent.pl on youtube*. URL: `https://www.youtube.com/watch?v=wo38aaH2F4E&spfreload=10` (visited on 02/21/2017).

[51] *Windows line breaks on Linux prevent removal of white space from end of line*. URL: `https://github.com/cmhughes/latexindent.pl/issues/256` (visited on 06/18/2021).

11.5 Contributors (in chronological order)

[1] Paulo Cereda. *arara rule, indent.yaml*. May 23, 2013. URL: `https://github.com/islandoftex/arara/blob/master/rules/arara-rule-indent.yaml` (visited on 03/19/2021).

[2] Harish Kumar. *Early version testing*. Nov. 10, 2013. URL: `https://github.com/harishkumarholla` (visited on 06/30/2017).

[3] Michel Voßkuhle. *Remove trailing white space*. Nov. 10, 2013. URL: `https://github.com/cmhughes/latexindent.pl/pull/12` (visited on 01/23/2017).

[4] Jacobo Diaz. *Changed shebang to make the script more portable*. July 23, 2014. URL: `https://github.com/cmhughes/latexindent.pl/pull/17` (visited on 01/23/2017).

[5] Jacobo Diaz. *Hiddenconfig*. July 21, 2014. URL: `https://github.com/cmhughes/latexindent.pl/pull/18` (visited on 01/23/2017).

[6] Jason Juang. *add in PATH installation*. Nov. 24, 2015. URL: `https://github.com/cmhughes/latexindent.pl/pull/38` (visited on 01/23/2017).

[7] mlep. *One sentence per line*. Aug. 16, 2017. URL: `https://github.com/cmhughes/latexindent.pl/issues/81` (visited on 01/08/2018).

[8] John Owens. *Paragraph line break routine removal*. May 27, 2017. URL: `https://github.com/cmhughes/latexindent.pl/issues/33` (visited on 05/27/2017).

[9] Cheng Xu (xu cheng). *always output log/help text to STDERR*. July 13, 2018. URL: `https://github.com/cmhughes/latexindent.pl/pull/121` (visited on 08/05/2018).

[10] Tom Zöhner (zoehneto). *Improving text wrap*. Feb. 4, 2018. URL: `https://github.com/cmhughes/latexindent.pl/issues/103` (visited on 08/04/2018).

[11] Sam Abey. *Print usage tip to STDERR only if STDIN is TTY*. Sept. 17, 2019. URL: `https://github.com/cmhughes/latexindent.pl/pull/174` (visited on 03/19/2021).

[12] Randolf J. *Alpine-linux instructions*. Aug. 10, 2020. URL: `https://github.com/cmhughes/latexindent.pl/pull/214` (visited on 08/10/2020).

[13] jeanlego. *Search localSettings in CWD as well*. Sept. 15, 2020. URL: `https://github.com/cmhughes/latexindent.pl/pull/221` (visited on 03/19/2021).

[14] newptcai. *Update appendices.tex*. Feb. 16, 2021. URL: `https://github.com/cmhughes/latexindent.pl/pull/221` (visited on 03/19/2021).

[15] qiancy98. *Locale encoding of file system*. May 6, 2021. URL: `https://github.com/cmhughes/latexindent.pl/pull/273` (visited on 05/06/2021).

[16] Alexander Regueiro. *Update help screen*. Mar. 18, 2021. URL: `https://github.com/cmhughes/latexindent.pl/pull/261` (visited on 03/19/2021).

[17] XuehaiPan. *-y switch upgrade*. Nov. 12, 2021. URL: `https://github.com/cmhughes/latexindent.pl/pull/297` (visited on 11/12/2021).

[18] XuehaiPan. *Verbatim block upgrade*. Oct. 3, 2021. URL: `https://github.com/cmhughes/latexindent.pl/pull/290` (visited on 10/03/2021).

[19] eggplants. *Add Dockerfile and its updater/releaser*. June 12, 2022. URL: `https://github.com/cmhughes/latexindent.pl/pull/370` (visited on 06/12/2022).

[20] Tom de Geus. *Adding Perl installation + pre-commit hook*. Jan. 21, 2022. URL: `https://github.com/cmhughes/latexindent.pl/pull/322` (visited on 01/21/2022).

[21] Jan Holthuis. *Fix pre-commit usage*. Mar. 31, 2022. URL: `https://github.com/cmhughes/latexindent.pl/pull/354` (visited on 04/02/2022).

[22] Nehctargl. *Added support for the XDG specification*. Dec. 23, 2022. URL: `https://github.com/cmhughes/latexindent.pl/pull/397` (visited on 12/23/2022).

[23] Junfeng Qiao. *Add w.r.t to betterFullStop*. May 25, 2023. URL: `https://github.com/cmhughes/latexindent.pl/pull/447` (visited on 05/25/2023).

[24] Henrik Sloot. *feat: add devcontainer configuration*. May 20, 2023. URL: `https://github.com/cmhughes/latexindent.pl/pull/443` (visited on 05/20/2023).

[25] Henrik Sloot. *fix: find local settings when working file dir is not working dir*. Feb. 15, 2023. URL: `https://github.com/cmhughes/latexindent.pl/pull/422` (visited on 02/15/2023).

[26] Jesse Stricker. *Create cruft directory if it does not exist*. July 12, 2023. URL: `https://github.com/cmhughes/latexindent.pl/pull/453` (visited on 07/12/2023).
[27] valtterikantanen. *fix: decode the name of the backup file*. Apr. 7, 2023. URL: `https://github.com/cmhughes/latexindent.pl/pull/439` (visited on 04/07/2023).
[28] fengzyf. *Encoding work*. June 15, 2024. URL: `https://github.com/cmhughes/latexindent.pl/pull/548` (visited on 06/15/2024).

SECTION A

Required Perl modules

If you intend to use `latexindent.pl` and *not* one of the supplied standalone executable files (`latexindent.exe` is available for Windows users without Perl, see Section 3.1.2), then you will need a few standard Perl modules.

If you can run the minimum code in Listing 597 as in

```
cmh:~$ perl helloworld.pl
```

then you will be able to run `latexindent.pl`, otherwise you may need to install the missing modules; see Sections A.1 and A.2.

LISTING 597: `helloworld.pl`

```
#!/usr/bin/perl

use strict;                         #     |
use warnings;                       #     |
use Encode;                         #     |
use Getopt::Long;                   #     |
use Data::Dumper;                   #  these modules are
use List::Util qw(max);             #  generally part
use PerlIO::encoding;               #  of a default perl distribution
use open ’:std’, ’:encoding(UTF-8)’;#     |
use Text::Wrap;                     #     |
use Text::Tabs;                     #     |
use FindBin;                        #     |
use File::Copy;                     #     |
use File::Basename;                 #     |
use File::Path;                     #     |
use File::HomeDir;                  # <--- typically requires install via cpanm
use YAML::Tiny;                     # <--- typically requires install via cpanm

print "hello␣world";
exit;
```

A.1 Module installer script

N: 2018-01-13

`latexindent.pl` ships with a helper script that will install any missing `perl` modules on your system; if you run

```
cmh:~$ perl latexindent-module-installer.pl
```

or

```
C:\Users\cmh>perl latexindent-module-installer.pl
```

then, once you have answered `Y`, the appropriate modules will be installed onto your distribution.

A.2 Manually installing modules

Manually installing the modules given in Listing 597 will vary depending on your operating system and `Perl` distribution.

A.2.1 Linux

A.2.1.1 perlbrew

Linux users may be interested in exploring Perlbrew [42]; an example installation would be:

```
cmh:~$ sudo apt-get install perlbrew
cmh:~$ perlbrew init
cmh:~$ perlbrew install perl-5.40.0
cmh:~$ perlbrew switch perl-5.40.0
cmh:~$ sudo apt-get install curl
cmh:~$ curl -L http://cpanmin.us | perl - App::cpanminus
cmh:~$ cpanm YAML::Tiny
cmh:~$ cpanm File::HomeDir
```

A.2.1.2 Ubuntu/Debian

For other distributions, the Ubuntu/Debian approach may work as follows

```
cmh:~$ sudo apt install perl
cmh:~$ sudo cpan -i App::cpanminus
cmh:~$ sudo cpanm YAML::Tiny
cmh:~$ sudo cpanm File::HomeDir
```

or else by running, for example,

```
cmh:~$ sudo perl -MCPAN -e’install␣"File::HomeDir"’
```

A.2.1.3 Ubuntu: using the texlive from apt-get

Ubuntu users that install texlive using `apt-get` as in the following

```
cmh:~$ sudo apt install texlive
cmh:~$ sudo apt install texlive-latex-recommended
```

may need the following additional command to work with `latexindent.pl`

```
cmh:~$ sudo apt install texlive-extra-utils
```

A.2.1.4 Ubuntu: users without perl

N: 2022-10-30

`latexindent-linux` is a standalone executable for Ubuntu Linux (and therefore does not require a Perl distribution) and caches copies of the Perl modules onto your system. It is available from [36].

A.2.1.5 Arch-based distributions

`latexindent` is included in Arch-packaged TeX Live, and can be installed by:

```
cmh:~$ sudo pacman -S texlive-binextra perl-yaml-tiny perl-file-homedir
```

To enable optional `-GCString` switch, install `perl-unicode-linebreak`:

```
cmh:~$ sudo pacman -S perl-unicode-linebreak
```

A.2.1.6 Alpine

If you are using Alpine, some `Perl` modules are not build-compatible with Alpine, but replacements are available through `apk`. For example, you might use the commands given in Listing 598; thanks to [12] for providing these details.

LISTING 598: `alpine-install.sh`

```
# Installing perl
apk --no-cache add miniperl perl-utils

# Installing incompatible latexindent perl dependencies via apk
apk --no-cache add \
    perl-log-dispatch \
    perl-namespace-autoclean \
    perl-specio \
    perl-unicode-linebreak

# Installing remaining latexindent perl dependencies via cpan
apk --no-cache add curl wget make
ls /usr/share/texmf-dist/scripts/latexindent
cd /usr/local/bin && \
    curl -L https://cpanmin.us/ -o cpanm && \
    chmod +x cpanm
cpanm -n App::cpanminus
cpanm -n File::HomeDir
cpanm -n Params::ValidationCompiler
cpanm -n YAML::Tiny
```

Users of NixOS might like to see https://github.com/cmhughes/latexindent.pl/issues/222 for tips.

A.2.2 Mac

Users of the Macintosh operating system might like to explore the following commands, for example:

```
cmh:~$ brew install perl
cmh:~$ brew install cpanm
cmh:~$
cmh:~$ cpanm YAML::Tiny
cmh:~$ cpanm File::HomeDir
```

Alternatively,

```
cmh:~$ brew install latexindent
```

N: 2022-10-30

`latexindent-macos` is a standalone executable for macOS (and therefore does not require a Perl distribution) and caches copies of the Perl modules onto your system. It is available from [36].

A.2.3 Windows

Strawberry Perl users on Windows might use `CPAN client`. All of the modules are readily available on CPAN [32]. `indent.log` will contain details of the location of the Perl modules on your system.

`latexindent.exe` is a standalone executable for Windows (and therefore does not require a Perl distribution) and caches copies of the Perl modules onto your system; if you wish to see where they are cached, use the `trace` option, e.g

```
C:\Users\cmh>latexindent.exe -t myfile.tex
```

A.3 The GCString switch

N: 2022-03-25

If you find that the `lookForAlignDelims` (as in Section 5.5) does not work correctly for your language, then you may wish to use the `Unicode::GCString` module .

This can be loaded by calling `latexindent.pl` with the `GCString` switch as in

```
cmh:~$ latexindent.pl --GCString myfile.tex
```

In this case, you will need to have the `Unicode::GCString` installed in your `perl` distribution by using, for example,

```
cmh:~$ cpanm Unicode::GCString
```

Note: this switch does *nothing* for `latexindent.exe` which loads the module by default. Users of `latexindent.exe` should not see any difference in behaviour whether they use this switch or not, as `latexindent.exe` loads the `Unicode::GCString` module.

SECTION B

Updating the path variable

`latexindent.pl` has a few scripts (available at [36]) that can update the `path` variables. Thank you to [6] for this feature. If you're on a Linux or Mac machine, then you'll want `CMakeLists.txt` from [36].

B.1 Add to path for Linux

To add `latexindent.pl` to the path for Linux, follow these steps:

1. download `latexindent.pl` and its associated modules, `defaultSettings.yaml`, to your chosen directory from [36] ;
2. within your directory, create a directory called `path-helper-files` and download `CMakeLists.txt` and `cmake_uninstall.cmake.in` from [36]/path-helper-files to this directory;
3. run

   ```
   cmh:~$ ls /usr/local/bin
   ```

   to see what is *currently* in there;
4. run the following commands

   ```
   cmh:~$ sudo apt-get update
   cmh:~$ sudo apt-get install --no-install-recommends cmake make # or any
       other generator
   cmh:~$ mkdir build && cd build
   cmh:~$ cmake ../path-helper-files
   cmh:~$ sudo make install
   ```

5. run

   ```
   cmh:~$ ls /usr/local/bin
   ```

   again to check that `latexindent.pl`, its modules and `defaultSettings.yaml` have been added.

To *remove* the files, run

```
cmh:~$ sudo make uninstall
```

B.2 Add to path for Windows

To add `latexindent.exe` to the path for Windows, follow these steps:

1. download `latexindent.exe`, `defaultSettings.yaml`, `add-to-path.bat` from [36] to your chosen directory;
2. open a command prompt and run the following command to see what is *currently* in your `%path%` variable;

```
C:\Users\cmh>echo %path%
```

3. right click on `add-to-path.bat` and *Run as administrator*;
4. log out, and log back in;
5. open a command prompt and run

   ```
   C:\Users\cmh>echo %path%
   ```

   to check that the appropriate directory has been added to your `%path%`.

To *remove* the directory from your `%path%`, run `remove-from-path.bat` as administrator.

Section C

Batches of files

N: 2022-03-25

You can instruct `latexindent.pl` to operate on multiple files. For example, the following calls are all valid

```
cmh:~$ latexindent.pl myfile1.tex
cmh:~$ latexindent.pl myfile1.tex myfile2.tex
cmh:~$ latexindent.pl myfile*.tex
```

We note the following features of the script in relation to the switches detailed in Section 3.

C.1 location of indent.log

If the `-c` switch is *not* active, then `indent.log` goes to the directory of the final file called.

If the `-c` switch *is* active, then `indent.log` goes to the specified directory.

C.2 interaction with -w switch

If the `-w` switch is active, as in

```
cmh:~$ latexindent.pl -w myfile*.tex
```

then files will be overwritten individually. Back-up files can be re-directed via the `-c` switch.

C.3 interaction with -o switch

If `latexindent.pl` is called using the `-o` switch as in

```
cmh:~$ latexindent.pl myfile*.tex -o=my-output-file.tex
```

and there are multiple files to operate upon, then the `-o` switch is ignored because there is only *one* output file specified.

More generally, if the `-o` switch does *not* have a `+` symbol at the beginning, then the `-o` switch will be ignored, and is turned it off.

For example

```
cmh:~$ latexindent.pl myfile*.tex -o=+myfile
```

will work fine because *each* `.tex` file will output to `<basename>myfile.tex`

Similarly,

```
cmh:~$ latexindent.pl myfile*.tex -o=++
```

will work because the 'existence check/incrementation' routine will be applied.

C.4 interaction with lines switch

This behaves as expected by attempting to operate on the line numbers specified for each file. See the examples in Section 8.

C.5 interaction with check switches

The exit codes for `latexindent.pl` are given in Table 1 on page 22.

When operating on multiple files with the check switch active, as in

```
cmh:~$ latexindent.pl myfile*.tex --check
```

then

- exit code 0 means that the text from *none* of the files has been changed;
- exit code 1 means that the text from *at least one* of the files been file changed.

The interaction with `checkv` switch is as in the check switch, but with verbose output.

C.6 when a file does not exist

What happens if one of the files can not be operated upon?

- if at least one of the files does not exist and `latexindent.pl` has been called to act upon multiple files, then the exit code is 3; note that `latexindent.pl` will try to operate on each file that it is called upon, and will not exit with a fatal message in this case;
- if at least one of the files can not be read and `latexindent.pl` has been called to act upon multiple files, then the exit code is 4; note that `latexindent.pl` will try to operate on each file that it is called upon, and will not exit with a fatal message in this case;
- if `latexindent.pl` has been told to operate on multiple files, and some do not exist and some cannot be read, then the exit code will be either 3 or 4, depending upon which it scenario it encountered most recently.

SECTION D

latexindent-yaml-schema.json

N: 2022-01-02

`latexindent.pl` ships with `latexindent-yaml-schema.json` which might help you when constructing your YAML files.

D.1 VSCode demonstration

To use `latexindent-yaml-schema.json` with `VSCode`, you can use the following steps:

1. download `latexindent-yaml-schema.json` from the `documentation` folder of [36], save it in whichever directory you would like, noting it for reference;
2. following the instructions from [37], for example, you should install the VSCode YAML extension [50];
3. set up your `settings.json` file using the directory you saved the file by adapting Listing 599; on my Ubuntu laptop this file lives at `/home/cmhughes/.config/Code/User/settings.json`.

LISTING 599: `settings.json`

```
{
  "yaml.schemas": {
    "/home/cmhughes/projects/latexindent/documentation/latexindent-yaml-schema.json":
    "/home/cmhughes/projects/latexindent/defaultSettings.yaml"
  },
  "redhat.telemetry.enabled": true
}
```

Alternatively, if you would prefer not to download the json file, you might be able to use an adapted version of Listing 600.

LISTING 600: `settings-alt.json`

```
{
  "yaml.schemas": {

     "https://raw.githubusercontent.com/cmhughes/latexindent.pl/main/documentation/latexindent-yaml-schema.json":
    "/home/cmhughes/projects/latexindent/defaultSettings.yaml"
  }
}
```

Finally, if your TeX distribution is up to date, then `latexindent-yaml-schema.json` *should* be in the documentation folder of your installation, so an adapted version of Listing 601 may work.

LISTING 601: `settings-alt1.json`

```
{
  "yaml.schemas": {
    "/usr/local/texlive/2021/texmf-dist/doc/support/latexindent/latexindent-yaml-schema.json":
    "/home/cmhughes/projects/latexindent/defaultSettings.yaml"
  }
}
```

If you have details of how to implement this schema in other editors, please feel encouraged to contribute to this documentation.

SECTION E

Using conda

If you use conda you'll only need

```
cmh:~$ conda install latexindent.pl -c conda-forge
```

This will install the executable and all its dependencies (including perl) in the activate environment. You don't even have to worry about `defaultSettings.yaml` as it included too, you can thus skip Sections A and B.

You can get a conda installation for example from [31] or from [30].

E.1 Sample conda installation on Ubuntu

On Ubuntu I followed the 64-bit installation instructions at [38] and then I ran the following commands:

```
cmh:~$ conda create -n latexindent.pl
cmh:~$ conda activate latexindent.pl
cmh:~$ conda install latexindent.pl -c conda-forge
cmh:~$ conda info --envs
cmh:~$ conda list
cmh:~$ conda run latexindent.pl -vv
```

I found the details given at [45] to be helpful.

SECTION F

Using docker

N: 2022-06-12 If you use docker you'll only need

```
cmh:~$ docker pull ghcr.io/cmhughes/latexindent.pl
```

This will download the image packed `latexindent`'s executable and its all dependencies. Thank you to [19] for contributing this feature; see also [40]. For reference, *ghcr* stands for *GitHub Container Repository*.

F.1 Sample docker installation on Ubuntu

To pull the image and show `latexindent`'s help on Ubuntu:

LISTING 602: `docker-install.sh`

```
# setup docker if not already installed
if ! command -v docker &> /dev/null; then
  sudo apt install docker.io -y
  sudo groupadd docker
  sudo gpasswd -a "$USER" docker
  sudo systemctl restart docker
  newgrp docker
fi

# download image and execute
docker pull ghcr.io/cmhughes/latexindent.pl
docker run ghcr.io/cmhughes/latexindent.pl -h
```

Once I have run the above, on subsequent logins I run

LISTING 603: `docker-install.sh`

```
newgrp docker
docker run ghcr.io/cmhughes/latexindent.pl -h
```

F.2 How to format on Docker

When you use `latexindent` with the docker image, you have to mount target `tex` file like this:

```
cmh:~$ docker run -v /path/to/local/myfile.tex:/myfile.tex
    ghcr.io/cmhughes/latexindent.pl -s -w myfile.tex
```

SECTION G

pre-commit

N: 2022-01-21

Users of `.git` may be interested in exploring the `pre-commit` tool [44], which is supported by `latexindent.pl`. Thank you to [20] for contributing this feature, and to [21] for their contribution to it.

To use the `pre-commit` tool, you will need to install `pre-commit`; sample instructions for Ubuntu are given in Section G.1. Once installed, there are two ways to use `pre-commit`: using `CPAN` or using `conda`, detailed in Section G.3 and Section G.4 respectively.

G.1 Sample pre-commit installation on Ubuntu

On Ubuntu I ran the following command:

```
cmh:~$ python3 -m pip install pre-commit
```

I then updated my path via .bashrc so that it includes the line in Listing 604.

LISTING 604: `.bashrc` update

```
...
export PATH=$PATH:/home/cmhughes/.local/bin
```

G.2 pre-commit defaults

The default values that are employed by `pre-commit` are shown in Listing 605.

LISTING 605: `.pre-commit-hooks.yaml` (default)

```
- id: latexindent
  name: latexindent.pl
  description: Run latexindent.pl (get dependencies using CPAN)
  minimum_pre_commit_version: 2.1.0
  entry: latexindent.pl
  args: ["--overwriteIfDifferent", "--silent", "--local"]
  language: perl
  types: [tex]
- id: latexindent-conda
  name: latexindent.pl
  description: Run latexindent.pl (get dependencies using Conda)
  minimum_pre_commit_version: 2.1.0
  entry: latexindent.pl
  args: ["--overwriteIfDifferent", "--silent", "--local"]
  language: conda
  types: [tex]
- id: latexindent-docker
  name: latexindent.pl
  description: Run latexindent.pl (get dependencies using Docker)
  minimum_pre_commit_version: 2.1.0
  entry: ghcr.io/cmhughes/latexindent.pl
  language: docker_image
  types: [tex]
  args: ["--overwriteIfDifferent", "--silent", "--local"]
```

In particular, the decision has deliberately been made (in collaboration with [21]) to have the default to employ the following switches: `overwriteIfDifferent`, `silent`, `local`; this is detailed in the lines that specify `args` in Listing 605.

Warning!

Users of `pre-commit` will, by default, have the `overwriteIfDifferent` switch employed. It is assumed that such users have version control in place, and are intending to overwrite their files.

G.3 pre-commit using CPAN

To use `latexindent.pl` with `pre-commit`, create the file `.pre-commit-config.yaml` given in Listing 606 in your git-repository.

LISTING 606: `.pre-commit-config.yaml` (cpan)

```
- repo: https://github.com/cmhughes/latexindent.pl
  rev: V3.24.5
  hooks:
  - id: latexindent
    args: [-s]
```

Once created, you should then be able to run the following command:

```
cmh:~$ pre-commit run --all-files
```

A few notes about Listing 606:

- the settings given in Listing 606 instruct `pre-commit` to use `CPAN` to get dependencies;
- this requires `pre-commit` and `perl` to be installed on your system;
- the `args` lists selected command-line options; the settings in Listing 606 are equivalent to calling

  ```
  cmh:~$ latexindent.pl -s myfile.tex
  ```

  for each `.tex` file in your repository;
- to instruct `latexindent.pl` to overwrite the files in your repository, then you can update Listing 606 so that `args: [-s, -w]`.

Naturally you can add options, or omit `-s` and `-w`, according to your preference.

G.4 pre-commit using conda

You can also rely on `conda` (detailed in Section E) instead of `CPAN` for all dependencies, including `latexindent.pl` itself.

LISTING 607: `.pre-commit-config.yaml` (conda)

```
- repo: https://github.com/cmhughes/latexindent.pl
  rev: V3.24.5
  hooks:
  - id: latexindent-conda
    args: [-s]
```

Once created, you should then be able to run the following command:

```
cmh:~$ pre-commit run --all-files
```

A few notes about Listing 606:

- the settings given in Listing 607 instruct `pre-commit` to use `conda` to get dependencies;
- this requires `pre-commit` and `conda` to be installed on your system;
- the `args` lists selected command-line options; the settings in Listing 606 are equivalent to calling

  ```
  cmh:~$ conda run latexindent.pl -s myfile.tex
  ```

  for each `.tex` file in your repository;
- to instruct `latexindent.pl` to overwrite the files in your repository, then you can update Listing 606 so that `args: [-s, -w]`.

G.5 pre-commit using docker

You can also rely on `docker` (detailed in Section F) instead of `CPAN` for all dependencies, including `latexindent.pl` itself.

LISTING 608: `.pre-commit-config.yaml` (docker)

```
- repo: https://github.com/cmhughes/latexindent.pl
  rev: V3.24.5
  hooks:
    - id: latexindent-docker
      args: [-s]
```

Once created, you should then be able to run the following command:

```
cmh:~$ pre-commit run --all-files
```

A few notes about Listing 606:

- the settings given in Listing 608 instruct `pre-commit` to use `docker` to get dependencies;
- this requires `pre-commit` and `docker` to be installed on your system;
- the `args` lists selected command-line options; the settings in Listing 606 are equivalent to calling

  ```
  cmh:~$ docker run -v /path/to/myfile.tex:/myfile.tex
      ghcr.io/cmhughes/latexindent.pl -s myfile.tex
  ```

  for each `.tex` file in your repository;
- to instruct `latexindent.pl` to overwrite the files in your repository, then you can update Listing 606 so that `args: [-s, -w]`.

G.6 pre-commit example using -l, -m switches

Let's consider a small example, with local `latexindent.pl` settings in `.latexindent.yaml`.

example 173 We use the local settings given in Listing 609.

LISTING 609: `.latexindent.yaml`

```
onlyOneBackUp: 1

modifyLineBreaks:
 oneSentencePerLine:
   manipulateSentences: 1
```

and `.pre-commit-config.yaml` as in Listing 610: ■

LISTING 610: `.pre-commit-config.yaml` (demo)

```yaml
- repo: https://github.com/cmhughes/latexindent.pl
  rev: V3.24.5
  hooks:
  - id: latexindent
    args: [-l, -m, -s, -w]
```

Now running

```
cmh:~$ pre-commit run --all-files
```

is equivalent to running

```
cmh:~$ latexindent.pl -l -m -s -w myfile.tex
```

for each `.tex` file in your repository.

A few notes about Listing 610:

- the `-l` option was added to use the local `.latexindent.yaml` (where it was specified to only create one back-up file, as `git` typically takes care of this when you use `pre-commit`);
- `-m` to modify line breaks; in addition to `-s` to suppress command-line output, and `-w` to format files in place.

■

SECTION H

indentconfig options

N: 2023-01-01

This section describes the possible locations for the main configuration file, discussed in Section 4. Thank you to [22] for this contribution.

The possible locations of `indentconfig.yaml` are read one after the other, and reading stops when a valid file is found in one of the paths.

Before stating the list, we give summarise in Table 5.

TABLE 5: indentconfig environment variable summaries

environment variable	type	Linux	macOS	Windows
LATEXINDENT_CONFIG	full path to file	✔	✔	✔
XDG_CONFIG_HOME	directory path	✔	✖	✖
LOCALAPPDATA	directory path	✖	✖	✔

The following list shows the checked options and is sorted by their respective priority. It uses capitalized and with a dollar symbol prefixed names (e.g. `$LATEXINDENT_CONFIG`) to symbolize environment variables. In addition to that the variable name `$homeDir` is used to symbolize your home directory.

1. The value of the environment variable `$LATEXINDENT_CONFIG` is treated as highest priority source for the path to the configuration file.
2. The next options are dependent on your operating system:
   - Linux:
     (a) The file at `$XDG_CONFIG_HOME/latexindent/indentconfig.yaml`
     (b) The file at `$homeDir/.config/latexindent/indentconfig.yaml`
   - Windows:
     (a) The file at `$LOCALAPPDATA\latexindent\indentconfig.yaml`
     (b) The file at `$homeDir\AppData\Local\latexindent\indentconfig.yaml`
   - Mac:
     (a) The file at `$homeDir/Library/Preferences/latexindent/indentconfig.yaml`
3. The file at `$homeDir/indentconfig.yaml`
4. The file at `$homeDir/.indentconfig.yaml`

H.1 Why to change the configuration location

This is mostly a question about what you prefer, some like to put all their configuration files in their home directory (see `$homeDir` above), whilst some like to sort their configuration. And if you don't care about it, you can just continue using the same defaults.

H.2 How to change the configuration location

This depends on your preferred location, if, for example, you would like to set a custom location, you would have to change the `$LATEXINDENT_CONFIG` environment variable.

Although the following example only covers `$LATEXINDENT_CONFIG`, the same process can be applied to `$XDG_CONFIG_HOME` or `$LOCALAPPDATA` because both are environment variables. You just have to change the path to your chosen configuration directory (e.g. `$homeDir/.config` or `$homeDir\AppData\Loca` on Linux or Windows respectively)

H.2.1 Linux

To change `$LATEXINDENT_CONFIG` on Linux you can run the following command in a terminal after changing the path:

```
cmh:~$ echo ’export␣LATEXINDENT_CONFIG="/home/cmh/latexindent-config.yaml"’ >> ~/.profile
```

Context: This command adds the given line to your `.profile` file (which is commonly stored in `$HOME/.profile`). All commands in this file a run after login, so the environment variable will be set after your next login.

You can check the value of `$LATEXINDENT_CONFIG` by typing

```
cmh:~$ echo $LATEXINDENT_CONFIG
cmh:~$ /home/cmh/latexindent-config.yaml
```

Linux users interested in `$XDG_CONFIG_HOME` can explore variations of the following commands

```
cmh:~$ echo $XDG_CONFIG_HOME
cmh:~$ echo ${XDG_CONFIG_HOME:=$HOME/.config}
cmh:~$ echo $XDG_CONFIG_HOME
cmh:~$ mkdir /home/cmh/.config/latexindent
cmh:~$ touch /home/cmh/.config/latexindent/indentconfig.yaml
```

H.2.2 Windows

To change `$LATEXINDENT_CONFIG` on Windows you can run the following command in `powershell.exe` after changing the path:

```
C:\Users\cmh>[Environment]::SetEnvironmentVariable
C:\Users\cmh>     ("LATEXINDENT_CONFIG", "\your\config\path", "User")
```

This sets the environment variable for every user session.

H.2.3 Mac

To change `$LATEXINDENT_CONFIG` on macOS you can run the following command in a terminal after changing the path:

```
cmh:~$ echo ’export␣LATEXINDENT_CONFIG="/your/config/path"’ >> ~/.profile
```

Context: This command adds the line to your `.profile` file (which is commonly stored in `$HOME/.profile`). All commands in this file a run after login, so the environment variable will be set after your next login.

SECTION I

paths demonstration

N: 2024-04-28

As detailed in Section 4.1 on page 23 , the `paths` field can be specified in any of your `YAML` files.

We will use the file in Listing 611 for demonstration in what follows.

LISTING 611: `paths-demo.tex`

```
\pathdemo[
opt arg
]{
mand arg
}
```

example 174 Consider the settings given in Listing 612 and Listing 613.

LISTING 612: `path1.yaml`

```
defaultIndent: ''
paths:
- path2.yaml
```

LISTING 613: `path2.yaml`

```
defaultIndent:␣'␣␣␣'
```

Upon calling

```
cmh:~$ latexindent.pl -l=path1.yaml paths-demo.tex
```

then we will receive the output given in Listing 614.

LISTING 614: `paths-demo-mod1.tex`

```
\pathdemo[
␣␣␣opt␣arg
]{
␣␣␣mand␣arg
}
```

We note that the settings from Listing 613 have been called from Listing 612.

On inspection of `indent.log` from the above call, we see the details of this part of the process given in Listing 615. ■

Listing 615: path-test1.txt

```
YAML settings, reading from the following files:
       Reading USER settings from path1.yaml
       Reading path information from path1.yaml
       ---
       defaultIndent: ''
       paths:
         - path2.yaml

       ---
       defaultIndent: ''
       paths:
         - path2.yaml

       Reading USER settings from path2.yaml
       ---
       defaultIndent: '   '
```

example 175 Consider the settings given in Listing 616 to Listing 618.

Listing 616: path3.yaml

```
defaultIndent: ''
paths:
- path4.yaml
```

Listing 617: path4.yaml

```
defaultIndent:␣'␣␣␣'
paths:
-␣path5.yaml
```

Listing 618: path5.yaml

```
defaultIndent:␣'␣'
```

Upon calling

```
cmh:~$ latexindent.pl -l=path3.yaml paths-demo.tex
```

then we will receive the output given in Listing 619.

Listing 619: paths-demo-mod3.tex

```
\pathdemo[
␣opt␣arg
]{
␣mand␣arg
}
```

We see that `path3.yaml` calls `path4.yaml` which in turn calls `path5.yaml`.

On inspection of `indent.log` from the above call, we see the details of this part of the process given in Listing 620.

Listing 620: `path-test3.txt`

```
YAML settings, reading from the following files:
       Reading USER settings from path3.yaml
       Reading path information from path3.yaml
       ---
       defaultIndent: ’’
       paths:
         - path4.yaml

       ---
       defaultIndent: ’’
       paths:
         - path4.yaml

       Reading USER settings from path4.yaml
       Reading path information from path4.yaml
       ---
       defaultIndent: ’   ’
       paths:
         - path5.yaml

       ---
       defaultIndent: ’   ’
       paths:
         - path5.yaml

       Reading USER settings from path5.yaml
       ---
       defaultIndent: ’ ’
```

SECTION J

logFilePreferences

Listing 36 on page 28 describes the options for customising the information given to the log file, and we provide a few demonstrations here.

example 176 Let's say that we start with the code given in Listing 621, and the settings specified in Listing 622.

LISTING 621: `simple.tex`

```
\begin{myenv}
  body of myenv
\end{myenv}
```

LISTING 622: `logfile-prefs1.yaml`

```
logFilePreferences:
    showDecorationStartCodeBlockTrace: "+++++"
    showDecorationFinishCodeBlockTrace: "-----"
```

If we run the following command (noting that `-t` is active)

```
cmh:~$ latexindent.pl -t -l=logfile-prefs1.yaml simple.tex
```

then on inspection of `indent.log` we will find the snippet given in Listing 623.

LISTING 623: `indent.log`

```
       +++++
TRACE: environment found: myenv
       No ancestors found for myenv
       Storing settings for myenvenvironments
       indentRulesGlobal specified (0) for environments, ...
       Using defaultIndent for myenv
       Putting linebreak after replacementText for myenv
       looking for COMMANDS and key = {value}
TRACE: Searching for commands with optional and/or mandatory arguments AND key =
    {value}
       looking for SPECIAL begin/end
TRACE: Searching myenv for special begin/end (see specialBeginEnd)
TRACE: Searching myenv for optional and mandatory arguments
       ... no arguments found
       -----
```

Notice that the information given about `myenv` is 'framed' using `+++++` and `-----` respectively. ■

SECTION K

Encoding

When using latexindent in different ways on different systems, the range of characters supported by its switches/flags/options (see Section 3.2 on page 15) may vary.

For the Windows executable file `latexindent.exe`, its options support UTF-8 characters.

For the Windows Perl script `latexindent.pl`, its option switch supports the characters supported by the encoding corresponding to the system code page. You can check the system code page by running the following command in either `cmd.exe` or `powershell.exe`:

```
C:\Users\cmh>  chcp
```

which may receive the following result

```
C:\Users\cmh>  Active code page: 936
```

and then the characters supported by the code page can be found in https://learn.microsoft.com/en-us/windows/win32/intl/code-page-identifiers. For example, the characters supported by the encoding corresponding to code page 936 are: ANSI/OEM Simplified Chinese (PRC, Singapore); Chinese Simplified (GB2312).

For Ubuntu Linux and macOS users, whether using the Perl script or the executable file, the options support UTF-8 characters.

SECTION L

dos2unix linebreak adjustment

`dos2unixlinebreaks`: ⟨*integer*⟩

N: 2021-06-19

If you use `latexindent.pl` on a dos-based Windows file on Linux then you may find that trailing horizontal space is not removed as you hope.

In such a case, you may wish to try setting `dos2unixlinebreaks` to 1 and employing, for example, the following command.

```
cmh:~$ latexindent.pl -y="dos2unixlinebreaks:1" myfile.tex
```

See [51] for further dertails.

SECTION M

Differences from Version 2.2 to 3.0

There are a few (small) changes to the interface when comparing Version 2.2 to Version 3.0. Explicitly, in previous versions you might have run, for example,

```
cmh:~$ latexindent.pl -o myfile.tex outputfile.tex
```

whereas in Version 3.0 you would run any of the following, for example,

```
cmh:~$ latexindent.pl -o=outputfile.tex myfile.tex
cmh:~$ latexindent.pl -o outputfile.tex myfile.tex
cmh:~$ latexindent.pl myfile.tex -o outputfile.tex
cmh:~$ latexindent.pl myfile.tex -o=outputfile.tex
cmh:~$ latexindent.pl myfile.tex -outputfile=outputfile.tex
cmh:~$ latexindent.pl myfile.tex -outputfile outputfile.tex
```

noting that the *output* file is given *next to* the `-o` switch.

The fields given in Listing 624 are *obsolete* from Version 3.0 onwards.

LISTING 624: Obsolete YAML fields from Version 3.0

```
alwaysLookforSplitBrackets
alwaysLookforSplitBrackets
checkunmatched
checkunmatchedELSE
checkunmatchedbracket
constructIfElseFi
```

There is a slight difference when specifying indentation after headings; specifically, we now write `indentAfterThisHeading` instead of `indent`. See Listings 625 and 626

LISTING 625: `indentAfterThisHeading` in Version 2.2

```
indentAfterHeadings:
    part:
       indent: 0
       level: 1
```

LISTING 626: `indentAfterThisHeading` in Version 3.0

```
indentAfterHeadings:
    part:
       indentAfterThisHeading: 0
       level: 1
```

To specify `noAdditionalIndent` for display-math environments in Version 2.2, you would write YAML as in Listing 627; as of Version 3.0, you would write YAML as in Listing 628 or, if you're using `-m` switch, Listing 629.

Listing 627: `noAdditionalIndent` in Version 2.2

```
noAdditionalIndent:
    \[: 0
    \]: 0
```

Listing 628: `noAdditionalIndent` for `displayMath` in Version 3.0

```
specialBeginEnd:
    displayMath:
        begin: ’\\\[’
        end: ’\\\]’
        lookForThis: 0
```

Listing 629: `noAdditionalIndent` for `displayMath` in Version 3.0

```
noAdditionalIndent:
    displayMath: 1
```

End

Listings

Index