latexindent.pl

Version 3.0

Chris Hughes

February 10, 2017

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 modifiy line breaks, and add comment symbols. All user options are customisable via the switches in the YAML interface.

Contents

and contributors! (See Section 7.2 on page 44.) For all communication, please visit [1].

Listings

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 [**cmhblog**] I received some positive feedback and follow-up feature requests. A big thank you to Harish Kumar 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 7.2 on page 44 for their valued contributions.

1.2 License

`latexindent.pl` is free and open source, and it always will be. 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 9) but you should still be careful when using it. The script has been tested on many files, but there are some known limitations (see **??**). 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 ([1]) with a complete minimum working example as I would like to improve the code as much as possible.

Before you try the script on anything important (like your thesis), test it out on the sample files in the `test-case` directory ([1]).

If you have used any version 2. of `latexindent.pl`, there are a few changes to the interface; see appendix C on page 47 and the comments throughout this document for details*

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 [**cmh:videodemo**]

FIX

As you look at Listings 1 to 6, 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 each user can personalize their indentation scheme.

In each of the samples given in Listings 1 to 6 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 1: `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 2: `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 3: `tikzset.tex`

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

LISTING 4: `tikzset.tex` default output

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

LISTING 5: `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 6: `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}}
```

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 and MiKTEX distributions for Windows users. These files are also avail-

able from github [1] should you wish to use them without a TEX distribution; in this case, you may like to read appendix B on page 45 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 3.1 and Section 3.2 respectively. We will discuss how to change the settings and behaviour of the script in Section 5 on page 8.

latexindent.pl ships with latexindent.exe for Windows users, so that you can use the script with or without a Perl distribution. If you plan to use latexindent.pl (i.e, the original Perl script) then you will need a few standard Perl modules – see appendix A on page 44 for details.

3.1 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.

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

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

-h, -help

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

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.

-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.

-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).

Note that using `-o` is equivalent to using

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

See appendix C on page 47 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` and if it is called with the `-l` switch and it finds `localSettings.yaml` in the same directory as `myfile.tex` then these settings will be

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

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 `localSettings.yaml`. In fact, you can specify *relative* path names to the current directory, but *not* absolute paths – for absolute paths, see Section 4 on page 6. Explicit demonstrations of how to use the `-l` switch are given throughout this documentation.

`-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`/`.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.

`-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.

`-g, -logfile`

```
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.

`-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 33

`latexindent.pl` can also be called on a file without the file extension, for example `latexindent.pl myfile` and in which case, you can specify the order in which extensions are searched for; see Listing 11 on page 9 for full details.

3.2 From `arara`

Using `latexindent.pl` from the command line is fine for some folks, but others may find it easier to use from `arara`. `arara` ships with a rule, `indent.yaml`, but in case you do not have this rule, you can find it at [**paulo**].

You can use the rule in any of the ways described in Listing 7 (or combinations thereof). In fact, `arara` allows yet greater flexibility – you can use `yes/no`, `true/false`, or `on/off` to toggle the various options.

LISTING 7: `arara` sample usage

```
% arara: indent
% arara: indent: {overwrite: yes}
% arara: indent: {output: myfile.tex}
% arara: indent: {silent: yes}
% arara: indent: {trace: yes}
% arara: indent: {localSettings: yes}
% arara: indent: {onlyDefault: on}
% arara: indent: { cruft: /home/cmhughes/Desktop }
% arara: indent: { modifylinebreaks: yes }
\documentclass{article}
...
```

Hopefully the use of these rules is fairly self-explanatory, but for completeness Table 1 shows the relationship between `arara` directive arguments and the switches given in Section 3.1.

TABLE 1: `arara` directive arguments and corresponding switches

`arara` directive argument	switch
`overwrite`	`-w`
`output`	`-o`
`silent`	`-s`
`trace`	`-t`
`localSettings`	`-l`
`onlyDefault`	`-d`
`cruft`	`-c`
`modifylinebreaks`	`-m`

The `cruft` directive does not work well when used with directories that contain spaces.

4 User, local settings, `indentconfig.yaml` and `.indentconfig.yaml`

Editing `defaultSettings.yaml` is not ideal as it may be overwritten when updating your distribution– a better way to customize the settings to your liking is to set up your own settings file, `mysettings.yaml` (or any name you like, provided it ends with `.yaml`). The only thing you have to do is tell `latexindent.pl` where to find it.

`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 [**jacobo-diaz-hidden-config**] for providing this feature. In what follows, we will use `indentconfig.yaml`, but it is understood that this equally represents `.indentconfig.yaml` as well. If you have both files in existence, `indentconfig.yaml` takes priority.

For Mac and Linux users, their home directory is `/username` while Windows (Vista onwards) is `C:\Users\username` [2] Listing 8 shows a sample `indentconfig.yaml` file.

[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 8: `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 that you write them in. 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 9 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.

LISTING 9: `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 recognize then you'll get a warning, otherwise you'll get confirmation that `latexindent.pl` has read your settings file [3].

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`.

4.1 `localSettings.yaml`

The `-l` switch tells `latexindent.pl` either to look for `localSettings.yaml` in the *same directory* as `myfile.tex`; alternatively, it may look for any other specified YAML file. Any settings file(s) specified in this way will be read *after* `defaultSettings.yaml` and, assuming they exist, user settings from `indentconfig.yaml`.

The *local* settings file may be called `localSettings.yaml`, and it can contain any switches that you'd like to change; a sample is shown in Listing 10.

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

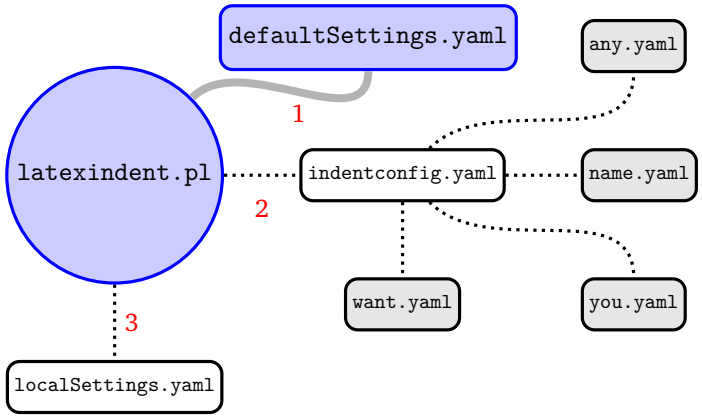


FIGURE 1: Schematic of the load order described in Section 4.2; solid lines represent mandatory files, dotted lines represent optional files. `indentconfig.yaml` can contain as many files as you like. The files will be loaded in order; if you specify settings for the same field in more than one file, the most recent takes priority.

LISTING 10: `localSettings.yaml` (example)

```
#  verbatim environments- environments specified
#  in this hash table will not be changed at all!
verbatimEnvironments:
cmhenvironment: 0
```

You can make sure that your local settings are loaded by checking `indent.log` for details; if `localSettings.yaml` can not be read then you will get a warning, otherwise you'll get confirmation that `latexindent.pl` has read `localSettings.yaml`.

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

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

4.2 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 4.1). You may specify relative paths to other YAML files using the `-l` switch, separating multiple files using commas.

A visual representation of this is given in Figure 1.

5 defaultSettings.yaml

`latexindent.pl` loads its settings from `defaultSettings.yaml` (rhymes with camel). 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.

`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.

Calling `latexindent.pl myfile` with the (default) settings specified in Listing 11 means that the script will first look for `myfile.tex`, then `myfile.sty`, `myfile.cls`, and finally `myfile.bib` in order[4].

LISTING 11: `fileExtensionPreference`

```
22 fileExtensionPreference:
23     .tex: 1
24     .sty: 2
25     .cls: 3
26     .bib: 4
```

`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`.

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.

`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
```

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

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

`logFilePreferences`: ⟨*fields*⟩

`latexindent.pl` writes information to `indent.log`, some of which can be customised by changing `logFilePreferences`; see Listing 12. If you load your own user settings (see Section 4 on page 6) 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. 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`.

LISTING 12: logFilePreferences

```
63 logFilePreferences:
64     showEveryYamlRead: 1
65     showAmalgamatedSettings: 0
66     endLogFileWith: '--------------'
67     showGitHubInfoFooter: 1
```

`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 13.

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`.

LISTING 13: verbatimEnvironments

```
71 verbatimEnvironments:
72     verbatim: 1
73     lstlisting: 1
```

`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 33).

LISTING 14: verbatimCommands

```
76 verbatimCommands:
77     verb: 1
78     lstinline: 1
```

`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 15.

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 16 for example.

LISTING 15: noIndentBlock

```
84 noIndentBlock:
85     noindent: 1
86     cmhtest: 1
```

LISTING 16: noIndentBlock demonstration

```
% \begin{noindent}
        this code
                won't
     be touched
                   by
              latexindent.pl!
%\end{noindent}
```

`removeTrailingWhitespace`: ⟨*fields*⟩

Trailing white space can be removed both *before* and *after* processing the document, as detailed in Listing 17; each of the fields can take the values 0 or 1. See Listings 184 to 186 on page 38 for before and after results. Thanks to [2] for providing this feature.

LISTING 17: removeTrailingWhitespace

```
89  removeTrailingWhitespace:
90      beforeProcessing: 0
91      afterProcessing: 1
```

`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 18. The behaviour of `latexindent.pl` on these environments is determined by their location (preamble or not), and the value `indentPreamble`, discussed next.

LISTING 18: fileContentsEnvironments

```
95  fileContentsEnvironments:
96      filecontents: 1
97      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 19, 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 19: lookForPreamble

```
103 lookForPreamble:
104     .tex: 1
105     .sty: 0
106     .cls: 0
107     .bib: 0
```

`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 20.

LISTING 20: Motivating `preambleCommandsBeforeEnvironments`

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

`defaultIndent`: ⟨*horizontal space*⟩

This is the default indentation (`\t` means a tab, and is the default value) used in the absence of other details for the command or environment we are working with; see `indentRules` in Section 5.2 on page 18 for more details.

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

`lookForAlignDelims`: ⟨*fields*⟩

This contains a list of environments and/or commands that are operated upon in a special way by `latexindent.pl` (see Listing 21). In fact, the fields in `lookForAlignDelims` can actually take two different forms: the *basic* version is shown in Listing 21 and the *advanced* version in Listing 24; we will discuss each in turn.

LISTING 21: `lookForAlignDelims` (basic)

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

The environments specified in this field will be operated on in a special way by `latexindent.pl`. In particular, it will try and align each column by its alignment tabs. It does have some limitations (discussed further in **??**), but in many cases it will produce results such as those in Listings 22 and 23.

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 15).

LISTING 22: `tabular1.tex`

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

LISTING 23: `tabular1.tex` default output

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

If you wish to remove the alignment of the `\\` within a delimiter-aligned block, then the advanced form of `lookForAlignDelims` shown in Listing 24 is for you.

LISTING 24: `tabular.yaml`

```
lookForAlignDelims:
   tabular:
      delims: 1
      alignDoubleBackSlash: 0
      spacesBeforeDoubleBackSlash: 0
   tabularx:
      delims: 1
   longtable: 1
```

Note that you can use a mixture of the basic and advanced form: in Listing 24 `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 up to 3 fields:

- `delims`: switch equivalent to simply specifying, for example, `tabular: 1` in the basic version shown in Listing 21 (default: 1);
- `alignDoubleBackSlash`: switch to determine if \\ should be aligned (default: 1);
- `spacesBeforeDoubleBackSlash`: optionally, specifies the number of spaces to be inserted before (non-aligned) \\. In order to use this field, `alignDoubleBackSlash` needs to be set to 0 (default: 0).

Assuming that you have the settings in Listing 24 saved in `tabular.yaml`, and the code from Listing 22 in `tabular1.tex` and you run

```
cmh:~$ latexindent.pl -l tabular.yaml tabular1.tex
```

then you should receive the before-and-after results shown in Listings 25 and 26; note that the ampersands have been aligned, but the \\ have not (compare the alignment of \\ in Listings 23 and 26).

LISTING 25: `tabular1.tex`

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

LISTING 26: `tabular1.tex` using Listing 24

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

Saving Listing 24 into `tabular1.yaml` as in Listing 28, and running the command

```
cmh:~$ latexindent.pl -l tabular1.yaml tabular1.tex
```

gives Listing 27; note the spacing before the \\.

LISTING 27: `tabular1.tex` using Listing 28

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

LISTING 28: `tabular1.yaml`

```
lookForAlignDelims:
   tabular:
      delims: 1
      alignDoubleBackSlash: 0
      spacesBeforeDoubleBackSlash: 3
   tabularx:
      delims: 1
   longtable: 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 14); for example, 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 -l matrix1.tex
```

then the before-and-after results shown in Listings 29 and 30 are achievable by default.

LISTING 29: matrix1.tex

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

LISTING 30: 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 31; the default output is shown in Listing 32. 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 31: align-block.tex

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

LISTING 32: align-block.tex default output

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

With reference to Table 2 on page 16 and the, yet undiscussed, fields of `noAdditionalIndent` and `indentRules` (see Section 5.2 on page 18), these comment-marked blocks are considered `environments`.

`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 34 and 35

LISTING 33: indentAfterItems

```
155 indentAfterItems:
156     itemize: 1
157     enumerate: 1
158     list: 1
```

LISTING 34: 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 35: 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 6 for details of how to configure user settings, and Listing 9 on page 7 in particular.)

LISTING 36: itemNames

```
164 itemNames:
165     item: 1
166     myitem: 1
```

`specialBeginEnd`: ⟨*fields*⟩

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 37 shows the default settings of `specialBeginEnd`.

LISTING 37: `specialBeginEnd`

```
specialBeginEnd:
    displayMath:
        begin: '\\\['
        end: '\\\]'
        lookForThis: 1
    inlineMath:
        begin: '(?<!\$)(?<!\\)\$(?!\$)'
        end: '(?<!\\)\$(?!\$)'
        lookForThis: 1
    displayMathTeX:
        begin: '\$\$'
        end: '\$\$'
        lookForThis: 1
```

The field `displayMath` represents `\[...\]`, `inlineMath` represents `$...$` and `displayMathTex` represents `$$...$$`. You can, of course, rename these in your own YAML files (see Section 4.1 on page 7); indeed, you might like to set up your own specil begin and end statements.

A demonstration of the before-and-after results are shown in Listings 38 and 39.

LISTING 38: `special1.tex` before

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

LISTING 39: `special1.tex` after

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

For each field, the `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`.

`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]

The default settings do *not* place indentation after a heading, but you can easily switch them on by changing `indentAfterThisHeading: 0` to `indentAfterThisHeading: 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

LISTING 40: `indentAfterHeadings`

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

[5] There is a slight difference in interface for this field when comparing Version 2.2 to Version 3.0; see appendix C on page 47 for details.

with `level: 3` because you do not want the indentation to go too 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 5.2 on the following page); you will find the default `indentRules` contains `chapter: " "` which tells `latexindent.pl` simply to use a space character after headings (once `indent` is set to 1 for `chapter`).

For example, assuming that you have read Section 4.1 on page 7, say that you have the code in Listing 41 saved into `headings1.yaml`, and that you have the text from Listing 42 saved into `headings1.tex`.

LISTING 41: `headings1.yaml`

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

LISTING 42: `headings1.tex`

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

If you run the command

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

then you should receive the output given in Listing 43.

LISTING 43: `headings1.tex` using Listing 41

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

LISTING 44: `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 41 so that the `paragraph level` is `1`; after running

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

you should now receive the code given in Listing 44; notice that the `paragraph` and `subsection` are at the same indentation level.

5.1 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.

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	`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 33 and 36 on page 14	`\begin{enumerate}`
		`  \item ...`
		`\end{enumerate}`

specialBeginEnd	User specified, see Listing 37 on page 15	`\[`
		`  ...`
		`\]`

afterHeading	User specified, see Listing 40 on page 15	`\chapter{title}`
		`  ...`
		`\section{title}`

filecontents	User specified, see Listing 18 on page 11 . . .

```
\begin{filecontents}
...
\end{filecontents}
```

We will refer to these code blocks in what follows.

5.2 noAdditionalIndent and indentRules

latexindent.pl operates on files by looking for code blocks, as detailed in Section 5.1 on page 16; for each type of code block in Table 2 on page 16 (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*⟩;
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 page 16; for reference, there follows a list of the code blocks covered.

5.2.1 Environments and their arguments

There are a few different YAML switches governing the indentation of environments; let's start with the simple sample code shown in Listing 53.

LISTING 53: myenv.tex

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

noAdditionalIndent: ⟨*fields*⟩

If we do not wish myenv to receive any additional indentation, we have a few choices available to us, as demonstrated in Listings 54 and 55.

LISTING 54: myenv-noAdd1.yaml

```
noAdditionalIndent:
    myenv: 1
```

LISTING 55: 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 56; note in particular that the environment myenv has not received any *additional* indentation, but that the outer environment *has* still received indentation.

LISTING 56: myenv.tex output (using either Listing 54 or Listing 55)

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

Upon changing the YAML files to those shown in Listings 57 and 58, 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 59.

LISTING 57: myenv-noAdd3.yaml

```
noAdditionalIndent:
    myenv: 0
```

LISTING 58: myenv-noAdd4.yaml

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

LISTING 59: myenv.tex output (using either Listing 57 or Listing 58)

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

Let's now allow myenv to have some optional and mandatory arguments, as in Listing 60.

LISTING 60: 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 61; 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 54), then *all* parts of the environment (body, optional and mandatory arguments) are assumed to want no additional indent.

LISTING 61: myenv-args.tex using Listing 54

```
\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}
```

We may customise `noAdditionalIndent` for optional and mandatory arguments of the `myenv` environment, as shown in, for example, Listings 62 and 63.

LISTING 62: myenv-noAdd5.yaml

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

LISTING 63: 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 64 and 65. Note that in Listing 64 the text for the *optional* argument has not received any additional indentation, and that in Listing 65 the *mandatory* argument has not received any additional indentation; in both cases, the *body* has not received any additional indentation.

LISTING 64: myenv-args.tex using Listing 62

```
\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 65: myenv-args.tex using Listing 63

```
\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*⟩

We may also specify indentation rules for environment code blocks using the `indentRules` field; see, for example, Listings 66 and 67.

LISTING 66: myenv-rules1.yaml

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

LISTING 67: 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 68; note in particular that the environment `myenv` has not received any *additional* indentation, but that the `outer` environment *has* still received indentation.

LISTING 68: myenv.tex output (using either Listing 66 or Listing 67)

```
\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.

Returning to the example in Listing 60 that contains optional and mandatory arguments. Upon using Listing 66 as in

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

we obtain the output in Listing 69; note that the body, optional argument and mandatory argument have *all* received the same customised indentation.

LISTING 69: myenv-args.tex using Listing 66

```
\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}
```

You can specify different indentation rules for the different features using, for example, Listings 70 and 71

LISTING 70: myenv-rules3.yaml

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

LISTING 71: 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 72 and 73.

LISTING 72: myenv-args.tex using Listing 70

```
\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 73: myenv-args.tex using Listing 71

```
\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 72, 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 73, 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 List-

LISTING 74: env-noAdditionalGlobal.yaml

```
noAdditionalIndentGlobal:
    environments: 0
```

ing 74). Let's say that you change the value of `environments` to 1 in Listing 74, 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 75 and 76; in Listing 75 notice that *both* environments receive no additional indentation but that the arguments of `myenv` still *do* receive indentation. In Listing 76 notice that the *outer* environment does not receive additional indentation, but because of the settings from `myenv-rules1.yaml` (in Listing 66 on page 21), the `myenv` environment still *does* receive indentation.

LISTING 75: `myenv-args.tex` using Listing 74

```
\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 76: `myenv-args.tex` using Listings 66 and 74

```
\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}
```

In fact, `noAdditionalIndentGlobal` also contains keys that control the indentation of optional and mandatory arguments; on referencing Listings 77 and 78

LISTING 77: `opt-args-no-add-glob.yaml`

```
noAdditionalIndentGlobal:
    optionalArguments: 1
```

LISTING 78: `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 79 and 80. Notice that in Listing 79 the *optional* argument has not received any additional indentation, and in Listing 80 the *mandatory* argument has not received any additional indentation.

LISTING 79: `myenv-args.tex` using Listing 77

```
\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 80: `myenv-args.tex` using Listing 78

```
\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 81; if you change the `environments` field to anything involving horizontal space, say `" "`, and then run the following commands

LISTING 81: `env-indentRulesGlobal.yaml`

```
indentRulesGlobal:
    environments: 0
```

```
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 82 and 83. Note that in Listing 82, both the environment blocks have received a single-space indentation, whereas in Listing 83 the `outer` environment has received single-space indentation (specified by `indentRulesGlobal`), but `myenv` has received `"    "`, as specified by the particular `indentRules` for `myenv` Listing 66 on page 21.

LISTING 82: `myenv-args.tex` using Listing 81

```
\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 83: `myenv-args.tex` using Listings 66 and 81

```
\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}
```

You can specify `indentRulesGlobal` for both optional and mandatory arguments, as detailed in Listings 84 and 85

LISTING 84: `opt-args-indent-rules-glob.yaml`

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

LISTING 85: `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 86 and 87. Note that the *optional* argument in Listing 86 has received two tabs worth of indentation, while the *mandatory* argument has done so in Listing 87.

LISTING 86: myenv-args.tex using Listing 84

```
\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 87: myenv-args.tex using Listing 85

```
\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}
```

5.2.2 Environments with items

With reference to Listings 33 and 36 on page 14, some commands may contain `item` commands; for the purposes of this discussion, we will use the code from Listing 34 on page 14.

Assuming that you've populated `itemNames` with the name of your `item`, you can put the item name into `noAdditionalIndent` as in Listing 88, although a more efficient approach may be to change the relevant field in `itemNames` to `0`. Similarly, you can customise the indentation that your `item` receives using `indentRules`, as in Listing 89

LISTING 88: item-noAdd1.yaml

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

LISTING 89: 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 90 and 91; note that in Listing 90 that the text after each `item` has not received any additional identation, and in Listing 91, the text after each `item` has received a single space of indentation, specified by Listing 89.

LISTING 90: items1.tex using Listing 88

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

LISTING 91: items1.tex using Listing 89

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

Alternatively, you might like to populate `noAdditionalIndentGlobal` or `indentRulesGlobal` using the `items` key, as demonstrated in Listings 92 and 93. Note that there is a need to 'reset/remove' the `item` field from `indentRules` in both cases (see the hierarchy description given on page 18) as the `item` command is a member of `indentRules` by default.

LISTING 92: items-noAdditionalGlobal.yaml

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

LISTING 93: 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 90 and 91 are obtained; note, however, that *all* such `item` commands without their own individual `noAdditionalIndent` or `indentRules` settings would behave as in these listings.

5.2.3 Commands with arguments

Let's begin with the simple example in Listing 94; when `latexindent.pl` operates on this file, the default output is shown in Listing 95.

LISTING 94: mycommand.tex

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

LISTING 95: 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 54 and 55 on page 18) we may specify `noAdditionalIndent` either in 'scalar' form, or in 'field' form, as shown in Listings 96 and 97

LISTING 96: mycommand-noAdd1.yaml

```
noAdditionalIndent:
    mycommand: 1
```

LISTING 97: 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 98 and 99

LISTING 98: mycommand.tex using Listing 96

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

LISTING 99: mycommand.tex using Listing 97

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

Note that in Listing 98 that the 'body', optional argument *and* mandatory argument have *all* received no additional indentation, while in Listing 99, 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.

We may further customise `noAdditionalIndent` for `mycommand` as we did in Listings 62 and 63 on page 20; explicit examples are given in Listings 100 and 101.

LISTING 100: mycommand-noAdd3.yaml

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

LISTING 101: 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 102 and 103.

LISTING 102: mycommand.tex using Listing 100

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

LISTING 103: mycommand.tex using Listing 101

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

Attentive readers will note that the body of `mycommand` in both Listings 102 and 103 has received no additional indent, even though `body` is explicitly set to `0` in both Listings 100 and 101. This is because, by default, `noAdditionalIndentGlobal` for `commands` is set to `1` by default; this can be easily fixed as in Listings 104 and 105.

LISTING 104: mycommand-noAdd5.yaml

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

LISTING 105: 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 106 and 107.

LISTING 106: mycommand.tex using Listing 104

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

LISTING 107: mycommand.tex using Listing 105

```
\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 70 and 71 on page 21 and Listings 81, 84 and 85 on page 23 and on page 24.

5.2.4 ifelsefi code blocks

Let's use the simple example shown in Listing 108; when `latexindent.pl` operates on this file, the output as in Listing 109; note that the body of each of the `\if` statements have been indented, and that the `\else` statement has been accounted for correctly.

LISTING 108: ifelsefi1.tex

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

LISTING 109: 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 110 and 111.

LISTING 110: ifnum-noAdd.yaml

```
noAdditionalIndent:
    ifnum: 1
```

LISTING 111: 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 112 and 113; note that in Listing 112, the `ifnum` code block has *not* received any additional indentation, while in Listing 113, the `ifnum` code block has received one tab and two spaces of indentation.

LISTING 112: ifelsefi1.tex using Listing 110

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

LISTING 113: ifelsefi1.tex using Listing 111

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

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 114 and 115.

LISTING 114: ifelsefi-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
    ifElseFi: 1
```

LISTING 115: 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 116 and 117; notice that in Listing 116 neither of the `ifelsefi` code blocks have received indentation, while in Listing 117 both code blocks have received a single space of indentation.

LISTING 116: ifelsefi1.tex using Listing 114

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

LISTING 117: ifelsefi1.tex using Listing 115

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

5.2.5 specialBeginEnd code blocks

Let's use the example from Listing 38 on page 15 which has default output shown in Listing 39 on page 15.

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 118 and 119.

LISTING 118: displayMath-noAdd.yaml

```
noAdditionalIndent:
    displayMath: 1
```

LISTING 119: 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 120 and 121; note that in Listing 120, the `displayMath` code block has *not* received any additional indentation, while in Listing 121, the `displayMath` code block has received three tabs worth of indentation.

LISTING 120: special1.tex using Listing 118

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

LISTING 121: special1.tex using Listing 119

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

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 122 and 123.

LISTING 122: special-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
    specialBeginEnd: 1
```

LISTING 123: 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 124 and 125; notice that in Listing 124 neither of the `special` code blocks have received indentation, while in Listing 125 both code blocks have received a single space of indentation.

LISTING 124: special1.tex using Listing 122

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

LISTING 125: special1.tex using Listing 123

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

5.2.6 `afterHeading` code blocks

Let's use the example Listing 126 for demonstration throughout this Section. As discussed on page 16, by default `latexindent.pl` will not add indentation after headings.

LISTING 126: headings2.tex

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

On using the YAML file in Listing 128 by running the command

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

we obtain the output in Listing 127. Note that the argument of `paragraph` has received (default) indentation, and that the body after the heading statement has received (default) indentation.

LISTING 127: headings2.tex using Listing 128

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

LISTING 128: headings3.yaml

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

If we specify `noAdditionalIndent` as in Listing 130 and run the command

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

then we receive the output in Listing 129. 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 129: `headings2.tex` using Listing 130

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

LISTING 130: `headings4.yaml`

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

Similarly, if we specify `indentRules` as in Listing 132 and run analogous commands to those above, we receive the output in Listing 131; note that the *body*, *mandatory argument* and content *after the heading* of `paragraph` have *all* received three tabs worth of indentation.

LISTING 131: `headings2.tex` using Listing 132

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

LISTING 132: `headings5.yaml`

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

We may, instead, specify `noAdditionalIndent` in 'field' form, as in Listing 134 which gives the output in Listing 133.

LISTING 133: `headings2.tex` using Listing 134

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

LISTING 134: `headings6.yaml`

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

Analogously, we may specify `indentRules` as in Listing 136 which gives the output in Listing 135; 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 135: `headings2.tex` using Listing 136

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

LISTING 136: `headings7.yaml`

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

Finally, let's consider `noAdditionalIndentGlobal` and `indentRulesGlobal` shown in Listings 138 and 138 respectively, with respective output in Listings 137 and 139. Note that in Listing 138 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 139, the *argument*

has received both a (default) tab plus two spaces of indentation (from the global rule specified in Listing 140), and the remaining body after `paragraph` has received just two spaces of indentation.

LISTING 137: `headings2.tex` using Listing 138

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

LISTING 138: `headings8.yaml`

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

LISTING 139: `headings2.tex` using Listing 140

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

LISTING 140: `headings9.yaml`

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

5.2.7 The remaining code blocks

Referencing the different types of code blocks in Table 2 on page 16, we have a few code blocks yet to cover; these are very similar to the `commands` code block type covered comprehensively in Section 5.2.3 on page 25, but a small discussion defining these remaining code blocks is necessary.

`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 16;
- then an = symbol;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

An example is shown in Listing 141, with the default output given in Listing 142.

LISTING 141: `pgfkeys1.tex`

```
\pgfkeys{/tikz/.cd,
start␣coordinate/.initial={0,
\vertfactor},
}
```

LISTING 142: `pgfkeys1.tex` default output

```
\pgfkeys{/tikz/.cd,
    ↦start␣coordinate/.initial={0,
    ↦    ↦    ↦\vertfactor},
}
```

In Listing 142, 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 18.

`namedGroupingBracesBrackets` This type of code block is mostly motivated by tikz-based code; we define this code block as follws:

- it must immediately follow either *horizontal space* OR *one or more line breaks* OR `{` OR `[` OR `$`;
- the name may contain the characters detailed in Table 2 on page 16;

- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

A simple example is given in Listing 143, with default output in Listing 144.

LISTING 143: child1.tex

```
\coordinate
child[grow=down]{
edge␣from␣parent␣[antiparticle]
node␣[above=3pt]␣{ $ C $ }
}
```

LISTING 144: 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`[6]. Referencing Listing 144, 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 18.

`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).

An example is shown in Listing 145 with default output give in Listing 146.

LISTING 145: psforeach1.tex

```
\psforeach{\row}{%
{
{3,2.8,2.7,3,3.1}},%
{2.8,1,1.2,2,3},%
}
```

LISTING 146: psforeach1.tex default output

```
\psforeach{\row}{%
    ┤{
    ┤   ┤   ┤{3,2.8,2.7,3,3.1}},%
    ┤{2.8,1,1.2,2,3},%
}
```

Referencing Listing 146, 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 18.

Users wishing to customise the mandatory and/or optional arguments on a *per-name* basis for the `UnNamedGroupingBracesBrackets` should use `always-un-named`.

`filecontents` code blocks behave just as `environments`, except that neither arguments nor items are sought.

5.2.8 Summary

Having considered all of the different types of code blocks, the functions of the fields given in Listings 147 and 148 should now make sense.

[6] You may like to verify this by using the `-tt` option and checking `indent.log`!

LISTING 147: noAdditionalIndentGlobal

```
noAdditionalIndentGlobal:
    environments: 0
    commands: 1
    optionalArguments: 0
    mandatoryArguments: 0
    ifElseFi: 0
    items: 0
    keyEqualsValuesBracesBrackets: 0
    namedGroupingBracesBrackets: 0
    UnNamedGroupingBracesBrackets: 0
    specialBeginEnd: 0
    afterHeading: 0
    filecontents: 0
```

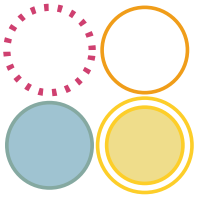
LISTING 148: indentRulesGlobal

```
indentRulesGlobal:
    environments: 0
    commands: 0
    optionalArguments: 0
    mandatoryArguments: 0
    ifElseFi: 0
    items: 0
    keyEqualsValuesBracesBrackets: 0
    namedGroupingBracesBrackets: 0
    UnNamedGroupingBracesBrackets: 0
    specialBeginEnd: 0
    afterHeading: 0
    filecontents: 0
```

6 The -m (modifylinebreaks) switch

All features described in this section will only be relevant if the -m switch is used.

`modifylinebreaks`: ⟨*fields*⟩



One of the most exciting features of Version 3.0 is 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 149.

LISTING 149: modifyLineBreaks -m

```
modifyLineBreaks:
    preserveBlankLines: 1
    condenseMultipleBlankLinesInto: 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:

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 below. 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`: ⟨*integer* ≥ 0⟩

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. As an example, Listing 150 shows a sample file with blank lines; upon running

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

the output is shown in Listing 151; note that the multiple blank lines have been condensed into one blank line, and note also that we have used the -m switch!

LISTING 150: mlb1.tex

```
before␣blank␣line


after␣blank␣line


after␣blank␣line
```

LISTING 151: mlb1.tex out output

```
before␣blank␣line

after␣blank␣line

after␣blank␣line
```

6.1 Poly-switches

Every other field in the `modifyLineBreaks` field uses poly-switches, and can take one of four integer values[7]:

−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>*.

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.

6.2 modifyLineBreaks for environments

We start by viewing a snippet of `defaultSettings.yaml` in Listing 152; note that it contains *global* settings (immediately after the `environments` field) and that *per-name* settings are also allowed – in the case of Listing 152, settings for `equation*` have been specified. Note that all poly-switches are *off* by default.

LISTING 152: environments -m

```
348     environments:
349         BeginStartsOnOwnLine: 0
350         BodyStartsOnOwnLine: 0
351         EndStartsOnOwnLine: 0
352         EndFinishesWithLineBreak: 0
353         equation*:
354             BeginStartsOnOwnLine: 0
355             BodyStartsOnOwnLine: 0
356             EndStartsOnOwnLine: 0
357             EndFinishesWithLineBreak: 0
```

6.2.1 Adding line breaks (poly-switches set to 1 or 2)

Let's begin with the simple example given in Listing 153; note that we have annotated key parts of the file using ♠, ♡, ♢ and ♣, these will be related to fields specified in Listing 152.

LISTING 153: env-mlb1.tex

```
before words♠ \begin{myenv}♡body of myenv♢\end{myenv}♣ after words
```

Let's explore `BeginStartsOnOwnLine` and `BodyStartsOnOwnLine` in Listings 154 and 155, and in particular, let's allow each of them in turn to take a value of 1.

[7] visual learners might like to associate one of the four circles in the logo with one of the four given values

LISTING 154: env-mlb1.yaml -m

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

LISTING 155: 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 156 and 157 respectively.

LISTING 156: env-mlb.tex using Listing 154

```
before␣words
\begin{myenv}body␣of␣myenv\end{myenv}␣after␣words
```

LISTING 157: env-mlb.tex using Listing 155

```
before␣words␣\begin{myenv}
    ♤body␣of␣myenv\end{myenv}␣after␣words
```

There are a couple of points to note:

- in Listing 156 a line break has been added at the point denoted by ♠ in Listing 153; no other line breaks have been changed;
- in Listing 157 a line break has been added at the point denoted by ♡ in Listing 153; furthermore, note that the *body* of myenv has received the appropriate (default) indentation.

Let's now change each of the 1 values in Listings 154 and 155 so that they are 2 and save them into env-mlb3.yaml and env-mlb4.yaml respectively (see Listings 158 and 159).

LISTING 158: env-mlb3.yaml -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 2
```

LISTING 159: env-mlb4.yaml -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 2
```

Upon running commands analogous to the above, we obtain Listings 160 and 161.

LISTING 160: env-mlb.tex using Listing 158

```
before␣words%
\begin{myenv}body␣of␣myenv\end{myenv}␣after␣words
```

LISTING 161: env-mlb.tex using Listing 159

```
before␣words␣\begin{myenv}%
    ♤body␣of␣myenv\end{myenv}␣after␣words
```

Note that line breaks have been added as in Listings 156 and 157, 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.

Let's explore EndStartsOnOwnLine and EndFinishesWithLineBreak in Listings 162 and 163, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 162: env-mlb5.yaml -m

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

LISTING 163: env-mlb6.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 1
```

After running the following commands,

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb5.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb6.yaml
```

the output is as in Listings 164 and 165.

LISTING 164: env-mlb.tex using Listing 162

```
before␣words␣\begin{myenv}body␣of␣myenv
\end{myenv}␣after␣words
```

LISTING 165: env-mlb.tex using Listing 163

```
before␣words␣\begin{myenv}body␣of␣myenv\end{myenv}
after␣words
```

There are a couple of points to note:

- in Listing 164 a line break has been added at the point denoted by ♢ in Listing 153 on page 35; no other line breaks have been changed and the `\end{myenv}` statement has *not* received indentation (as intended);
- in Listing 165 a line break has been added at the point denoted by ♣ in Listing 153 on page 35.

Let's now change each of the `1` values in Listings 162 and 163 so that they are 2 and save them into `env-mlb7.yaml` and `env-mlb8.yaml` respectively (see Listings 166 and 167).

LISTING 166: env-mlb7.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 2
```

LISTING 167: env-mlb8.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 2
```

Upon running commands analogous to the above, we obtain Listings 168 and 169.

LISTING 168: env-mlb.tex using Listing 166

```
before␣words␣\begin{myenv}body␣of␣myenv%
\end{myenv}␣after␣words
```

LISTING 169: env-mlb.tex using Listing 167

```
before␣words␣\begin{myenv}body␣of␣myenv\end{myenv}%
after␣words
```

Note that line breaks have been added as in Listings 164 and 165, 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.

If you ask `latexindent.pl` to add a line break (possibly with a comment) using a poly-switch value of 1 (or 2), it will only do so if necessary. For example, if you process the file in Listing 170 on page 36 using any of the YAML files presented so far in this section, it will be left unchanged.

LISTING 170: env-mlb2.tex

```
before␣words
\begin{myenv}
␣␣body␣of␣myenv
\end{myenv}
after␣words
```

LISTING 171: env-mlb3.tex

```
before␣words
\begin{myenv}␣␣%
␣␣body␣of␣myenv%
\end{myenv}%␣
after␣words
```

In contrast, the output from processing the file in Listing 171 will vary depending on the poly-switches used; in Listing 172 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 173 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 171 and by setting the other poly-switches considered so far to 2 in turn.

LISTING 172: env-mlb3.tex using Listing 155 on page 35

```
before␣words
\begin{myenv}
    ↵%
    ↵body␣of␣myenv%
\end{myenv}%␣
after␣words
```

LISTING 173: env-mlb3.tex using Listing 159 on page 35

```
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 152 on page 34, an example is shown for the `equation*` environment.

6.2.2 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. We will consider the example code given in Listing 174, noting in particular the positions of the line break highlighters, ♠, ♡, ♢ and ♣, together with the associated YAML files in Listings 175 to 178.

LISTING 174: env-mlb4.tex

```
before words♠
\begin{myenv}♡
body of myenv♢
\end{myenv}♣
after words
```

LISTING 175: env-mlb9.yaml -m

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

LISTING 176: env-mlb10.yaml -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: -1
```

LISTING 177: env-mlb11.yaml -m

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

LISTING 178: env-mlb12.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: -1
```

After running the commands

```
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb9.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb10.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb11.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb12.yaml
```

we obtain the respective output in Listings 179 to 182.

LISTING 179: env-mlb4.tex using Listing 175

```
before␣words\begin{myenv}
    ♃body␣of␣myenv
\end{myenv}
after␣words
```

LISTING 180: env-mlb4.tex using Listing 176

```
before␣words
\begin{myenv}body␣of␣myenv
\end{myenv}
after␣words
```

LISTING 181: env-mlb4.tex using Listing 177

```
before␣words
\begin{myenv}
    ♃body␣of␣myenv\end{myenv}
after␣words
```

LISTING 182: env-mlb4.tex using Listing 178

```
before␣words
\begin{myenv}
    ♃body␣of␣myenv
\end{myenv}after␣words
```

Notice that in

- Listing 179 the line break denoted by ♠ in Listing 174 has been removed;
- Listing 180 the line break denoted by ♡ in Listing 174 has been removed;
- Listing 181 the line break denoted by ♢ in Listing 174 has been removed;
- Listing 182 the line break denoted by ♣ in Listing 174 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 175 to 178 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-mlb9.yaml,env-mlb10.yaml,env-mlb11.yaml,env-mlb12.yaml
```

which gives the output in Listing 153 on page 35.

About trailing horizontal space Recall that on page 11 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; let's consider the file shown in Listing 183, which highlights trailing spaces.

LISTING 183: `env-mlb5.tex`

```
before␣words␣␣␣♠
\begin{myenv}␣␣␣␣␣␣␣␣␣␣␣♡
body␣of␣myenv␣␣␣␣␣␣♢
\end{myenv}␣␣␣␣␣♣
after␣words
```

LISTING 184: `removeTWS-before.yaml`

```
removeTrailingWhitespace:
    beforeProcessing: 1
```

The output from the following commands

```
cmh:~$ latexindent.pl -m env-mlb5.tex -l env-mlb9.yaml,env-mlb10.yaml,env-mlb11.yaml,env-mlb12.yaml
cmh:~$ latexindent.pl -m env-mlb5.tex -l
    env-mlb9.yaml,env-mlb10.yaml,env-mlb11.yaml,env-mlb12.yaml,removeTWS-before.yaml
```

is shown, respectively, in Listings 185 and 186; note that the trailing horizontal white space has been preserved (by default) in Listing 185, while in Listing 186, it has been removed using the switch specified in Listing 184.

LISTING 185: `env-mlb5.tex` using Listings 179 to 182

```
before␣words␣␣␣\begin{myenv}␣␣␣␣␣␣␣␣␣␣␣body␣of␣myenv␣␣␣␣␣␣\end{myenv}␣␣␣␣␣after␣words
```

LISTING 186: `env-mlb5.tex` using Listings 179 to 182 *and* Listing 184

```
before␣words\begin{myenv}body␣of␣myenv\end{myenv}after␣words
```

Blank lines Now let's consider the file in Listing 187, which contains blank lines.

LISTING 187: `env-mlb6.tex`

```
before words♠

\begin{myenv}♡

body of myenv♢

\end{myenv}♣

after words
```

LISTING 188: `UnpreserveBlankLines.yaml` `-m`

```
modifyLineBreaks:
    preserveBlankLines: 0
```

Upon running the following commands

```
cmh:~$ latexindent.pl -m env-mlb6.tex -l env-mlb9.yaml,env-mlb10.yaml,env-mlb11.yaml,env-mlb12.yaml
cmh:~$ latexindent.pl -m env-mlb6.tex -l
    env-mlb9.yaml,env-mlb10.yaml,env-mlb11.yaml,env-mlb12.yaml,UnpreserveBlankLines.yaml
```

we receive the respective outputs in Listings 189 and 190. In Listing 189 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 190, we have allowed the poly-switches to remove blank lines because, in Listing 188, we have set `preserveBlankLines` to 0.

LISTING 189: `env-mlb6.tex` using Listings 179 to 182

```
before␣words

\begin{myenv}

    body␣of␣myenv

\end{myenv}

after␣words
```

LISTING 190: `env-mlb6.tex` using Listings 179 to 182 *and* Listing 188

```
before␣words\begin{myenv}body␣of␣myenv\end{myenv}after␣words
```

6.3 Poly-switches for other code blocks

Rather than repeat the examples shown for the environment code blocks (in Section 6.2 on page 34), 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. Note also that, by design, line breaks involving `verbatim`, `filecontents` and 'comment-marked' code blocks (Listing 31 on page 14) can *not* be modified using `latexindent.pl`.

TABLE 3: Poly-switch mappings for all code-block types

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 statement★ \else□ body of else statement◇ \fi♣ after words	♠ IfStartsOnOwnLine ♡ BodyStartsOnOwnLine ★ ElseStartsOnOwnLine □ ElseFinishesWithLineBreak ◇ FiStartsOnOwnLine ♣ FiFinishesWithLineBreak

optionalArguments	`...`♠ `[`♡ `body of opt arg`♢ `]`♣ `...`	♠ LSqBStartsOnOwnLine[8] ♡ OptArgBodyStartsOnOwnLine ♢ RSqBStartsOnOwnLine ♣ RSqBFinishesWithLineBreak
mandatoryArguments	`...`♠ `{`♡ `body of mand arg`♢ `}`♣ `...`	♠ LCuBStartsOnOwnLine[9] ♡ MandArgBodyStartsOnOwnLine ♢ RCuBStartsOnOwnLine ♣ RCuBFinishesWithLineBreak
commands	`before words`♠ `\mycommand`♡ ⟨*arguments*⟩	♠ CommandStartsOnOwnLine ♡ CommandNameFinishesWithLineBreak
namedGroupingBraces Brackets	`before words`♠ `myname`♡ ⟨*braces/brackets*⟩	♠ NameStartsOnOwnLine ♡ NameFinishesWithLineBreak
keyEqualsValuesBraces Brackets	`before words`♠ `key`•`=`♡ ⟨*braces/brackets*⟩	♠ KeyStartsOnOwnLine • EqualsStartsOnOwnLine ♡ EqualsFinishesWithLineBreak
items	`before words`♠ `\item`♡ `...`	♠ ItemStartsOnOwnLine ♡ ItemFinishesWithLineBreak
specialBeginEnd	`before words`♠ `\[`♡ `body of special`♢ `\]`♣ `after words`	♠ SpecialBeginStartsOnOwnLine ♡ SpecialBodyStartsOnOwnLine ♢ SpecialEndStartsOnOwnLine ♣ SpecialEndFinishesWithLineBreak

6.4 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 page 39) and `LCuBStartsOnOwnLine` for mandatory arguments, and `LSqBStartsOnOwnLine` for optional arguments.

[8] LSqB stands for Left Square Bracket
[9] LCuB stands for Left Curly Brace

Let's begin with the code in Listing 200 and the YAML settings in Listing 202; with reference to Table 3 on the following page, the key `CommandNameFinishesWithLineBreak` is an alias for `BodyStartsOnOwnLine`.

LISTING 200: `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 201; note that the *second* mandatory argument beginning brace `{` has had its leading line break removed, but that the *first* brace has not.

LISTING 201: `mycommand1.tex` using Listing 202

```
\mycommand
{
    mand␣arg␣text
    mand␣arg␣text}{
    mand␣arg␣text
    mand␣arg␣text}
```

LISTING 202: `mycom-mlb1.yaml` -m

```
modifyLineBreaks:
    commands:
        CommandNameFinishesWithLineBreak: 0
    mandatoryArguments:
        LCuBStartsOnOwnLine: -1
```

Now let's change the YAML file so that it is as in Listing 204; upon running the analogous command to that given above, we obtain Listing 203; both beginning braces `{` have had their leading line breaks removed.

LISTING 203: `mycommand1.tex` using Listing 204

```
\mycommand{
    mand␣arg␣text
    mand␣arg␣text}{
    mand␣arg␣text
    mand␣arg␣text}
```

LISTING 204: `mycom-mlb2.yaml` -m

```
modifyLineBreaks:
    commands:
        CommandNameFinishesWithLineBreak: -1
    mandatoryArguments:
        LCuBStartsOnOwnLine: -1
```

Now let's change the YAML file so that it is as in Listing 206; upon running the analogous command to that given above, we obtain Listing 205.

LISTING 205: `mycommand1.tex` using Listing 206

```
\mycommand
{
    mand␣arg␣text
    mand␣arg␣text}
{
    mand␣arg␣text
    mand␣arg␣text}
```

LISTING 206: `mycom-mlb3.yaml` -m

```
modifyLineBreaks:
    commands:
        CommandNameFinishesWithLineBreak: -1
    mandatoryArguments:
        LCuBStartsOnOwnLine: 1
```

6.5 Conflicting poly-switches: sequential code blocks

It is very easy to have conflicting poly-switches; if we use the example from Listing 200 on page 41, and consider the YAML settings given in Listing 208. The output from running

```
cmh:~$ latexindent.pl -m -l=mycom-mlb4.yaml mycommand1.tex
```

is given in Listing 208.

LISTING 207: mycommand1.tex using Listing 208

```
\mycommand
{
    mand arg text
    mand arg text}{
    mand arg text
    mand arg text}
```

LISTING 208: mycom-mlb4.yaml -m

```
modifyLineBreaks:
    mandatoryArguments:
        LCuBStartsOnOwnLine: -1
        RCuBFinishesWithLineBreak: 1
```

Studying Listing 208, 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 207, 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 prioirty.

We can explore this further by considering the YAML settings in Listing 210; upon runnning the command

```
cmh:~$ latexindent.pl -m -l=mycom-mlb5.yaml mycommand1.tex
```

we obtain the output given in Listing 209.

LISTING 209: mycommand1.tex using Listing 210

```
\mycommand
{
    mand arg text
    mand arg text}
{
    mand arg text
    mand arg text}
```

LISTING 210: mycom-mlb5.yaml -m

```
modifyLineBreaks:
    mandatoryArguments:
        LCuBStartsOnOwnLine: 1
        RCuBFinishesWithLineBreak: -1
```

As previously, the most-recently-processed code block takes prioirty – as before, the second (i.e, *last*) argument. Exploring this further, we consider the YAML settings in Listing 212, which give associated output in Listing 211.

LISTING 211: mycommand1.tex using Listing 212

```
\mycommand
{
    mand arg text
    mand arg text}%
{
    mand arg text
    mand arg text}
```

LISTING 212: 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.

6.6 Conflicting poly-switches: nested code blocks

Now let's consider an example when nested code blocks have conflicting poly-switches; we'll use the code in Listing 213, noting that it contains nested environments.

LISTING 213: `nested-env.tex`

```
\begin{one}
one␣text
\begin{two}
two␣text
\end{two}
\end{one}
```

Let's use the YAML settings given in Listing 215, which upon running the command

```
cmh:~$ latexindent.pl -m -l=nested-env-mlb1.yaml nested-env.tex
```

gives the output in Listing 215.

LISTING 214: `nested-env.tex` using Listing 215

```
\begin{one}
    ┤one␣text
    ┤\begin{two}
    ┤    ┤two␣text\end{two}\end{one}
```

LISTING 215: `nested-env-mlb1.yaml` `-m`

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

In Listing 215, 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 213, 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 215, 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, and then 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}).

We can explore this further using the poly-switches in Listing 217, which give the output in Listing 217; upon running the command

```
cmh:~$ latexindent.pl -m -l=nested-env-mlb2.yaml nested-env.tex
```

we obtain the output given in Listing 217.

LISTING 216: nested-env.tex using Listing 217

```
\begin{one}
    one text
    \begin{two}
        two text
    \end{two}\end{one}
```

LISTING 217: 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.

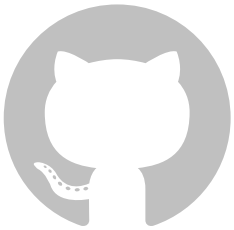
The indentation is done in Phase 2, and then in Phase 3, the one environment is found and processed first, followed by the two environment. *At this stage*, the two environment finds EndFinishesWithLineBreak is −1, so it removes the trailing line break; remember, at this point, latexindent.pl has completely finished with the one environment.

7 References

7.1 External links

[1] *Home of latexindent.pl*. URL: https://github.com/cmhughes/latexindent.pl (visited on 01/23/2017).

7.2 Contributors



[2] Michel Voßkuhle. *Remove trailing white space*. Nov. 10, 2013. URL: https://github.com/cmhughes/latexindent.pl/pull/12 (visited on 01/23/2017).

A Required Perl modules

If you intend to use latexindent.pl and *not* one of the supplied standalone executable files, then you will need a few standard Perl modules – if you can run the minimum code in Listing 218 (perl helloworld.pl) then you will be able to run latexindent.pl, otherwise you may need to install the missing modules.

LISTING 218: helloworld.pl

```
#!/usr/bin/perl

use strict;
use warnings;
use FindBin;
use YAML::Tiny;
use File::Copy;
use File::Basename;
use Getopt::Long;
use File::HomeDir;

print "hello world";
exit;
```

My default installation on Ubuntu 12.04 did *not* come with all of these modules as standard, but Strawberry Perl for Windows [**strawberryperl**] did.

Installing the modules given in Listing 218 will vary depending on your operating system and `Perl` distribution. For example, Ubuntu users might visit the software center, or else run

```
cmh:~$ sudo perl -MCPAN -e 'install "File::HomeDir"'
```

Linux users may be interested in exploring Perlbrew [**perlbrew**]; possible installation and setup options follow for Ubuntu (other distributions will need slightly different commands).

```
cmh:~$ sudo apt-get install perlbrew
cmh:~$ perlbrew install perl-5.20.1
cmh:~$ perlbrew switch perl-5.20.1
cmh:~$ sudo apt-get install curl
cmh:~$ curl -L http://cpanmin.us | perl - App::cpanminus
cmh:~$ cpanm YAML::Tiny
cmh:~$ cpanm File::HomeDir
```

Strawberry Perl users on Windows might use `CPAN client`. All of the modules are readily available on CPAN [**cpan**].

`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
```

B Updating the `path` variable

`latexindent.pl` has a few scripts (available at [1]) that can update the `path` variables [10]. If you're on a Linux or Mac machine, then you'll want `CMakeLists.txt` from [1].

FIX

B.1 Add to path for Linux

To add `latexindent.pl` to the path for Linux, follow these steps:

[10] Thanks to [**jasjuang**] for this feature!

1. download `latexindent.pl` and its associated modules, `defaultSettings.yaml`, to your chosen directory from [1] ;
2. within your directory, create a directory called `path-helper-files` and download `CMakeLists.txt` and `cmake_uninstall.cmake.in` from [1]/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 install cmake
cmh:~$ sudo apt-get update && sudo apt-get install build-essential
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` 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 [1] to your chosen directory;
2. open a command prompt and run 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.

C 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 219 are *obsolete* from Version 3.0 onwards.

LISTING 219: 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 220 and 221

LISTING 220: `indentAfterThisHeading` in Version 2.2

```
indentAfterHeadings:
    part:
       indent: 0
       level: 1
```

LISTING 221: `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 222; as of Version 3.0, you would write YAML as in Listing 223 or, if you're using `-m` switch, Listing 224.

LISTING 222: `noAdditionalIndent` in Version 2.2

```
noAdditionalIndent:
    \[: 0
    \]: 0
```

LISTING 223: `noAdditionalIndent` for `displayMath` in Version 3.0

```
specialBeginEnd:
    displayMath:
        begin: '\\\['
        end: '\\\]'
        lookForThis: 0
```

LISTING 224: `noAdditionalIndent` for `displayMath` in Version 3.0

```
noAdditionalIndent:
    displayMath: 1
```