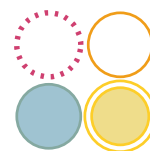


latexindent.pl

Version 3.2.1



Chris Hughes *

June 25, 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 modify line breaks, and add comment symbols. All user options are customisable via the switches in the YAML interface.

Contents

1	Introduction	5
1.1	Thanks	5
1.2	License	5
2	Demonstration: before and after	5
3	How to use the script	6
3.1	From the command line	6
3.2	From <code>arara</code>	11
4	User, local settings, <code>indentconfig.yaml</code> and <code>.indentconfig.yaml</code>	11
4.1	<code>localSettings.yaml</code>	12
4.2	Settings load order	13
5	<code>defaultSettings.yaml</code>	13
5.1	The code blocks known <code>latexindent.pl</code>	23
5.2	<code>noAdditionalIndent</code> and <code>indentRules</code>	24
5.2.1	Environments and their arguments	25
5.2.2	Environments with items	31
5.2.3	Commands with arguments	32
5.2.4	<code>ifelsefi</code> code blocks	34
5.2.5	<code>specialBeginEnd</code> code blocks	36
5.2.6	<code>afterHeading</code> code blocks	37
5.2.7	The remaining code blocks	39
5.2.8	Summary	40
5.3	Commands and the strings between their arguments	41
6	The <code>-m (modifylinebreaks)</code> switch	43
6.1	Poly-switches	53
6.2	<code>modifyLineBreaks</code> for environments	54
6.2.1	Adding line breaks (poly-switches set to 1 or 2)	54

*and contributors! See Section 8.2 on page 65. For all communication, please visit [6].



6.2.2 Removing line breaks (poly-switches set to <code>-1</code>)	56
6.3 Poly-switches for other code blocks	59
6.4 Partnering <code>BodyStartsOnOwnLine</code> with argument-based poly-switches	60
6.5 Conflicting poly-switches: sequential code blocks	61
6.6 Conflicting poly-switches: nested code blocks	63
7 Conclusions and known limitations	64
8 References	65
8.1 External links	65
8.2 Contributors	65
A Required Perl modules	65
B Updating the path variable	66
B.1 Add to path for Linux	66
B.2 Add to path for Windows	67
C Differences from Version 2.2 to 3.0	67

Listings

LISTING 1: <code>filecontents1.tex</code>	6	LISTING 35: <code>matrix1.tex</code>	20
LISTING 2: <code>filecontents1.tex</code> default output	6	LISTING 36: <code>matrix1.tex</code> default output	20
LISTING 3: <code>tikzset.tex</code>	6	LISTING 37: <code>align-block.tex</code>	20
LISTING 4: <code>tikzset.tex</code> default output	6	LISTING 38: <code>align-block.tex</code> default output	20
LISTING 5: <code>pstricks.tex</code>	6	LISTING 39: <code>indentAfterItems</code>	20
LISTING 6: <code>pstricks.tex</code> default output	6	LISTING 40: <code>items1.tex</code>	21
LISTING 7: <code>arara</code> sample usage	11	LISTING 41: <code>items1.tex</code> default output	21
LISTING 11: <code>fileExtensionPreference</code>	14	LISTING 42: <code>itemName</code>	21
LISTING 12: <code>logFilePreferences</code>	15	LISTING 43: <code>specialBeginEnd</code>	21
LISTING 13: <code>verbatimEnvironments</code>	15	LISTING 44: <code>special1.tex</code> before	22
LISTING 14: <code>verbatimCommands</code>	15	LISTING 45: <code>special1.tex</code> default output	22
LISTING 15: <code>noIndentBlock</code>	15	LISTING 46: <code>indentAfterHeadings</code>	22
LISTING 16: <code>noIndentBlock</code> demonstration	16	LISTING 47: <code>headings1.yaml</code>	22
LISTING 17: <code>removeTrailingWhitespace</code>	16	LISTING 48: <code>headings1.tex</code>	22
LISTING 18: <code>fileContentsEnvironments</code>	16	LISTING 49: <code>headings1.tex</code> using Listing 47	23
LISTING 19: <code>lookForPreamble</code>	16	LISTING 50: <code>headings1.tex</code> second modification	23
LISTING 20: <code>Motivating preambleCommandsBeforeEnvironments</code>	17	LISTING 59: <code>myenv.tex</code>	25
LISTING 22: <code>tabular1.tex</code>	17	LISTING 60: <code>myenv-noAdd1.yaml</code>	25
LISTING 23: <code>tabular1.tex</code> default output	17	LISTING 61: <code>myenv-noAdd2.yaml</code>	25
LISTING 24: <code>tabular.yaml</code>	17	LISTING 62: <code>myenv.tex</code> output (using either Listing 60 or Listing 61)	25
LISTING 25: <code>tabular1.tex</code>	18	LISTING 63: <code>myenv-noAdd3.yaml</code>	26
LISTING 26: <code>tabular1.tex</code> using Listing 24	18	LISTING 64: <code>myenv-noAdd4.yaml</code>	26
LISTING 27: <code>tabular1.tex</code> using Listing 28	18	LISTING 65: <code>myenv.tex</code> output (using either Listing 63 or Listing 64)	26
LISTING 28: <code>tabular1.yaml</code>	18	LISTING 66: <code>myenv-args.tex</code>	26
LISTING 29: <code>tabular2.tex</code>	19	LISTING 67: <code>myenv-args.tex</code> using Listing 60	26
LISTING 30: <code>tabular2.yaml</code>	19	LISTING 68: <code>myenv-noAdd5.yaml</code>	27
LISTING 31: <code>tabular3.yaml</code>	19	LISTING 69: <code>myenv-noAdd6.yaml</code>	27
LISTING 32: <code>tabular2-default.tex</code>	19	LISTING 70: <code>myenv-args.tex</code> using Listing 68	27
LISTING 33: <code>tabular2-mc.tex</code>	19	LISTING 71: <code>myenv-args.tex</code> using Listing 69	27
LISTING 34: <code>tabular2-no-max.tex</code>	19		



LISTING 72: myenv-rules1.yaml	27	LISTING 119: ifelsefi1.tex using Listing 117.....	35
LISTING 73: myenv-rules2.yaml	27	LISTING 120: ifelsefi-noAdd-glob.yaml	35
LISTING 74: myenv.tex output (using either Listing 72 or Listing 73).....	28	LISTING 121: ifelsefi-indent-rules-global.yaml 35	
LISTING 75: myenv-args.tex using Listing 72.....	28	LISTING 122: ifelsefi1.tex using Listing 120.....	36
LISTING 76: myenv-rules3.yaml	28	LISTING 123: ifelsefi1.tex using Listing 121.....	36
LISTING 77: myenv-rules4.yaml	28	LISTING 124: displayMath-noAdd.yaml	36
LISTING 78: myenv-args.tex using Listing 76.....	29	LISTING 125: displayMath-indent-rules.yaml.....	36
LISTING 79: myenv-args.tex using Listing 77.....	29	LISTING 126: special1.tex using Listing 124.....	36
LISTING 80: noAdditionalIndentGlobal	29	LISTING 127: special1.tex using Listing 125.....	36
LISTING 81: myenv-args.tex using Listing 80.....	29	LISTING 128: special-noAdd-glob.yaml	36
LISTING 82: myenv-args.tex using Listings 72 and 80.....	29	LISTING 129: special-indent-rules-global.yaml 36	
LISTING 83: opt-args-no-add-glob.yaml	30	LISTING 130: special1.tex using Listing 128.....	37
LISTING 84: mand-args-no-add-glob.yaml	30	LISTING 131: special1.tex using Listing 129.....	37
LISTING 85: myenv-args.tex using Listing 83.....	30	LISTING 132: headings2.tex	37
LISTING 86: myenv-args.tex using Listing 84.....	30	LISTING 133: headings2.tex using Listing 134.....	37
LISTING 87: indentRulesGlobal	30	LISTING 134: headings3.yaml	37
LISTING 88: myenv-args.tex using Listing 87.....	31	LISTING 135: headings2.tex using Listing 136.....	37
LISTING 89: myenv-args.tex using Listings 72 and 87.....	31	LISTING 136: headings4.yaml	37
LISTING 90: opt-args-indent-rules-glob.yaml.....	31	LISTING 137: headings2.tex using Listing 138.....	38
LISTING 91: mand-args-indent-rules-glob.yaml ..	31	LISTING 138: headings5.yaml	38
LISTING 92: myenv-args.tex using Listing 90.....	31	LISTING 139: headings2.tex using Listing 140.....	38
LISTING 93: myenv-args.tex using Listing 91.....	31	LISTING 140: headings6.yaml	38
LISTING 94: item-noAdd1.yaml	32	LISTING 141: headings2.tex using Listing 142.....	38
LISTING 95: item-rules1.yaml	32	LISTING 142: headings7.yaml	38
LISTING 96: items1.tex using Listing 94	32	LISTING 143: headings2.tex using Listing 144.....	38
LISTING 97: items1.tex using Listing 95	32	LISTING 144: headings8.yaml	38
LISTING 98: items-noAdditionalGlobal.yaml.....	32	LISTING 145: headings2.tex using Listing 146.....	39
LISTING 99: items-indentRulesGlobal.yaml.....	32	LISTING 146: headings9.yaml	39
LISTING 100: mycommand.tex	33	LISTING 147: pgfkeys1.tex	39
LISTING 101: mycommand.tex default output	33	LISTING 148: pgfkeys1.tex default output	39
LISTING 102: mycommand-noAdd1.yaml	33	LISTING 149: child1.tex	39
LISTING 103: mycommand-noAdd2.yaml	33	LISTING 150: child1.tex default output	39
LISTING 104: mycommand.tex using Listing 102.....	33	LISTING 151: psforeach1.tex	40
LISTING 105: mycommand.tex using Listing 103.....	33	LISTING 152: psforeach1.tex default output	40
LISTING 106: mycommand-noAdd3.yaml	33	LISTING 153: noAdditionalIndentGlobal	41
LISTING 107: mycommand-noAdd4.yaml	33	LISTING 154: indentRulesGlobal.....	41
LISTING 108: mycommand.tex using Listing 106.....	34	LISTING 155: commandCodeBlocks.....	41
LISTING 109: mycommand.tex using Listing 107.....	34	LISTING 156: pstricks1.tex	41
LISTING 110: mycommand-noAdd5.yaml	34	LISTING 157: pstricks1 default output	41
LISTING 111: mycommand-noAdd6.yaml	34	LISTING 158: pstricks1.tex using Listing 159.....	42
LISTING 112: mycommand.tex using Listing 110.....	34	LISTING 159: noRoundParentheses.yaml	42
LISTING 113: mycommand.tex using Listing 111.....	34	LISTING 160: pstricks1.tex using Listing 161.....	42
LISTING 114: ifelsefi1.tex	35	LISTING 161: defFunction.yaml.....	42
LISTING 115: ifelsefi1.tex default output	35	LISTING 162: tikz-node1.tex	42
LISTING 116: ifnum-noAdd.yaml	35	LISTING 163: tikz-node1 default output	42
LISTING 117: ifnum-indent-rules.yaml	35	LISTING 164: tikz-node1.tex using Listing 165.....	43
LISTING 118: ifelsefi1.tex using Listing 116.....	35	LISTING 165: draw.yaml	43
		LISTING 166: tikz-node1.tex using Listing 167.....	43



LISTING 167: no-to.yaml	43	LISTING 216: env-mlb.tex using Listing 214.....	55
LISTING 168: modifyLineBreaks	44	LISTING 217: env-mlb5.yaml	55
LISTING 169: mlb1.tex	44	LISTING 218: env-mlb6.yaml	55
LISTING 170: mlb1.tex out output	44	LISTING 219: env-mlb.tex using Listing 217.....	55
LISTING 171: textWrapOptions	45	LISTING 220: env-mlb.tex using Listing 218.....	55
LISTING 172: textwrap1.tex	45	LISTING 221: env-mlb7.yaml	55
LISTING 173: textwrap1-mod1.tex	45	LISTING 222: env-mlb8.yaml	55
LISTING 174: textwrap1.yaml	45	LISTING 223: env-mlb.tex using Listing 221.....	55
LISTING 175: textwrap2.tex	45	LISTING 224: env-mlb.tex using Listing 222.....	55
LISTING 176: textwrap2-mod1.tex	46	LISTING 225: env-mlb2.tex	56
LISTING 177: textwrap3.tex	46	LISTING 226: env-mlb3.tex	56
LISTING 178: textwrap3-mod1.tex	46	LISTING 227: env-mlb3.tex using Listing 210 on page 54	56
LISTING 179: textWrapOptions	46	LISTING 228: env-mlb3.tex using Listing 214 on page 55	56
LISTING 180: textwrap4.tex	46	LISTING 229: env-mlb4.tex	57
LISTING 181: textwrap4-mod2.tex	47	LISTING 230: env-mlb9.yaml	57
LISTING 182: textwrap2.yaml	47	LISTING 231: env-mlb10.yaml	57
LISTING 183: removeParagraphLineBreaks	47	LISTING 232: env-mlb11.yaml	57
LISTING 184: shortlines.tex	48	LISTING 233: env-mlb12.yaml	57
LISTING 185: remove-para1.yaml	48	LISTING 234: env-mlb4.tex using Listing 230.....	57
LISTING 186: shortlines1.tex	48	LISTING 235: env-mlb4.tex using Listing 231.....	57
LISTING 187: shortlines1-tws.tex	48	LISTING 236: env-mlb4.tex using Listing 232.....	57
LISTING 188: shortlines-mand.tex	49	LISTING 237: env-mlb4.tex using Listing 233.....	57
LISTING 189: shortlines-opt.tex	49	LISTING 238: env-mlb5.tex	58
LISTING 190: shortlines-mand1.tex	49	LISTING 239: removeTWS-before.yaml	58
LISTING 191: shortlines-opt1.tex	49	LISTING 240: env-mlb5.tex using Listings 234 to 237.....	58
LISTING 192: shortlines-envs.tex	50	LISTING 241: env-mlb5.tex using Listings 234 to 237 and Listing 239	58
LISTING 193: remove-para2.yaml	50	LISTING 242: env-mlb6.tex	58
LISTING 194: remove-para3.yaml	50	LISTING 243: UnpreserveBlankLines.yaml	58
LISTING 195: shortlines-envs2.tex	50	LISTING 244: env-mlb6.tex using Listings 234 to 237.....	59
LISTING 196: shortlines-envs3.tex	51	LISTING 245: env-mlb6.tex using Listings 234 to 237 and Listing 243	59
LISTING 197: shortlines-md.tex	51	LISTING 255: mycommand1.tex	61
LISTING 198: remove-para4.yaml	51	LISTING 256: mycommand1.tex using Listing 257.....	61
LISTING 199: shortlines-md4.tex	52	LISTING 257: mycom-mlb1.yaml	61
LISTING 200: paragraphsStopAt	52	LISTING 258: mycommand1.tex using Listing 259.....	61
LISTING 201: sl-stop.tex	52	LISTING 259: mycom-mlb2.yaml	61
LISTING 202: stop-command.yaml	52	LISTING 260: mycommand1.tex using Listing 261.....	61
LISTING 203: stop-comment.yaml	52	LISTING 261: mycom-mlb3.yaml	61
LISTING 204: sl-stop4.tex	53	LISTING 262: mycommand1.tex using Listing 263.....	62
LISTING 205: sl-stop4-command.tex	53	LISTING 263: mycom-mlb4.yaml	62
LISTING 206: sl-stop4-comment.tex	53	LISTING 264: mycommand1.tex using Listing 265.....	62
LISTING 207: environments	54	LISTING 265: mycom-mlb5.yaml	62
LISTING 208: env-mlb1.tex	54	LISTING 266: mycommand1.tex using Listing 267.....	62
LISTING 209: env-mlb1.yaml	54	LISTING 267: mycom-mlb6.yaml	62
LISTING 210: env-mlb2.yaml	54	LISTING 268: nested-env.tex	63
LISTING 211: env-mlb.tex using Listing 209.....	54	LISTING 269: nested-env.tex using Listing 269.....	63
LISTING 212: env-mlb.tex using Listing 210.....	54	LISTING 270: nested-env-mlb1.yaml	63
LISTING 213: env-mlb3.yaml	55		
LISTING 214: env-mlb4.yaml	55		
LISTING 215: env-mlb.tex using Listing 213.....	55		



LISTING 271: `nested-env.tex` using Listing 272..... 64
 LISTING 272: `nested-env-mlb2.yaml` 64

LISTING 273: `helloworld.pl` 65

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 T_EX stack exchange [1] 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 8.2 on page 65 for their valued contributions, and thank you to those who report bugs and request features at [6].

1.2 License

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

Before you start using it on any important files, bear in mind that `latexindent.pl` has the option to overwrite your `.tex` files. It will always make at least one backup (you can choose how many it makes, see page 14) but you should still be careful when using it. The script has been tested on many files, but there are some known limitations (see Section 7). You, the user, are responsible for ensuring that you maintain backups of your files before running `latexindent.pl` on them. I think it is important at this stage to restate an important part of the license here:

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

There is certainly no malicious intent in releasing this script, and I do hope that it works as you expect it to; if it does not, please first of all make sure that you have the correct settings, and then feel free to let me know at [6] 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 [6].

If you have used any version 2. of `latexindent.pl`, there are a few changes to the interface; see appendix C on page 67 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 [12]

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 you can personalize your 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="StrawberryPerl",
url="http://strawberryperl.com/"}
@online{cmhblog,
title="APerlscript...
url="...
}
\end{filecontents}
```

LISTING 3: tikzset.tex

```
\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 2: filecontents1.tex default output

```
\begin{filecontents}{mybib.bib}
% @online{strawberryperl,
% title="StrawberryPerl",
% url="http://strawberryperl.com/"}
% @online{cmhblog,
% title="APerlscript...
% url="...
% }
\end{filecontents}
```

LISTING 4: tikzset.tex default output

```
\tikzset{
% shrink_inner_sep/.code={
% % \pgfkeysgetvalue...
% % \pgfkeysgetvalue...
% % }
% }
}
```

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 T_EXLive distribution for Linux and Mac users; latexindent.exe ships as part of the T_EXLive and MiK_TE_X distributions for Windows users. These files are also available from github [6] should you wish to use them without a T_EX distribution; in this case, you may like to read appendix B on page 66 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 13.

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 65 for details.

As you read through this documentation, you'll occasionally see dates shown in the margin (for example, next to this paragraph!) which detail the date of the version in which the feature was implemented; the 'N' stands for 'new as of the date shown'. If you see a ★, it means that the feature is new as of the release of the current version.

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 customized, 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.

-v, -version

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

This will output only the version number to the terminal.

-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 exists¹. 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` as above is equivalent to using

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



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

N: 2017-06-25 ★

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

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

N: 2017-06-25 ★

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

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

N: 2017-06-25 ★

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

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

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

Calling `latexindent.pl` with simply

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

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

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

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

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

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

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

See appendix C on page 67 for details of how the interface has changed from Version 2.2 to Version 3.0 for this flag.

`-s`, `-silent`

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

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



`-t, -trace`

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

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

`-tt, -ttrace`

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

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

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

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

`latexindent.pl` will always load `defaultSettings.yaml` (rhymes with camel) and if it is called with the `-l` switch and it finds `localSettings.yaml` in the same directory as `myfile.tex` then these settings will be added to the indentation scheme. Information will be given in `indent.log` on the success or failure of loading `localSettings.yaml`.

The `-l` flag can take an *optional* parameter which details the name (or names separated by commas) of a YAML file(s) that resides in the same directory as `myfile.tex`; you can use this option if you would like to load a settings file in the current working directory that is *not* called `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 11. Explicit demonstrations of how to use the `-l` switch are given throughout this documentation.

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

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

which translate, respectively, to

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

Note that the following is *not* allowed:

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



and

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

will *only* load `localSettings.yaml`, and `myyaml.yaml` will be ignored.

You may also choose to omit the `yaml` extension, such as

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

-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 43

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

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

and in which case, you can specify the order in which extensions are searched for; see Listing 11 on page 14 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 [2].

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 }
\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
<code>overwrite</code>	<code>-w</code>
<code>output</code>	<code>-o</code>
<code>silent</code>	<code>-s</code>
<code>trace</code>	<code>-t</code>
<code>localSettings</code>	<code>-l</code>
<code>onlyDefault</code>	<code>-d</code>
<code>cruft</code>	<code>-c</code>
<code>modifylinebreaks</code>	<code>-m</code>

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 [5] 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.

²If 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 to look for localSettings.yaml in the *same directory* as myfile.tex. 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
```

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

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

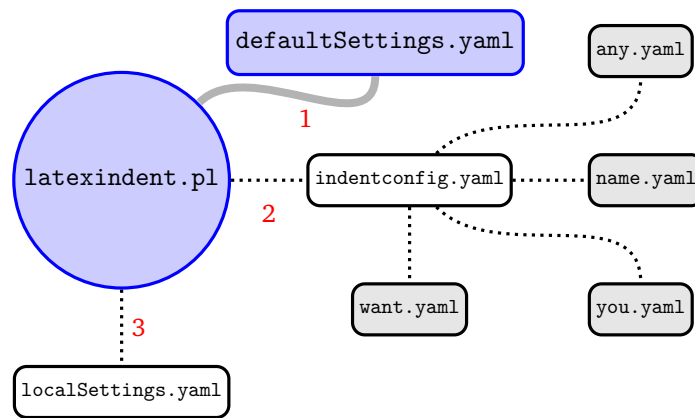


FIGURE 1: Schematic of the load order described in Section 4.2; solid lines represent mandatory files, dotted lines represent optional files. `indentconfig.yaml` can contain as many files as you like. The files will be loaded in order; if you specify settings for the same field in more than one file,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

Your settings file can contain any switches that you’d like to change; a sample is shown in Listing 10, and you’ll find plenty of further examples throughout this manual.

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

4.2 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`. 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 order⁴.

LISTING 11:
fileExtensionPreference

```
fileExtensionPreference:
  .tex: 1
  .sty: 2
  .cls: 3
  .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` for the first time.

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

onlyOneBackUp: *<integer>*

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

maxNumberOfBackUps: *<integer>*

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

cycleThroughBackUps: *<integer>*

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

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

The default value of `cycleThroughBackUps` is 0.

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



`logFilePreferences:` *<fields>*

latexindent.pl writes information to indent.log, some of which can be customized by changing logFilePreferences; see Listing 12. If you load your own user settings (see Section 4 on page 11) 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

```
logFilePreferences:
  showEveryYamlRead: 1
  showAmalgamatedSettings: 0
  endLogFileWith: '-----'
  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.

LISTING 13:
verbatimEnvironments

```
87 verbatimEnvironments:
88   verbatim: 1
89   lstlisting: 1
90   minted: 1
```

LISTING 14:
verbatimCommands

```
93 verbatimCommands:
94   verb: 1
95   lstinline: 1
```

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

`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 43).

`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

```
101 noIndentBlock:
102   noindent: 1
103   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 239 to 241 on page 58 for before and after results. Thanks to [13] for providing this feature.

`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.

`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.

`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 17:
removeTrailingWhitespace:
106 removeTrailingWhitespace:
107   beforeProcessing: 0
108   afterProcessing: 1
```

```
LISTING 18:
fileContentsEnvironments
112 fileContentsEnvironments:
113   filecontents: 1
114   filecontents*: 1
```

```
LISTING 19:
lookForPreamble
120 lookForPreamble:
121   .tex: 1
122   .sty: 0
123   .cls: 0
124   .bib: 0
```



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 24 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.

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 Section 7), 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 on page 15).

LISTING 22: tabular1.tex

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

LISTING 21:
lookForAlignDelims
(basic)

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

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
    multiColumnGrouping: 0
    alignRowsWithoutMaxDelims: 1
  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 any of the following 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).
- `multiColumnGrouping`: details if `latexindent.pl` should group columns underneath a `\multicolumn` command (default: 0);
- `alignRowsWithoutMaxDelims`: details if rows that do not contain the maximum number of delimiters should be formatted so as to have the ampersands aligned (default: 1).

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
```

Now consider the file `tabular2.tex` in Listing 29, which contains a `\multicolumn` command, and the YAML files in Listings 30 and 31.



LISTING 29: tabular2.tex

```

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

```

LISTING 30: tabular2.yaml

```

lookForAlignDelims:
  tabular:
    multiColumnGrouping: 1

```

LISTING 31: tabular3.yaml

```

lookForAlignDelims:
  tabular:
    alignRowsWithoutMaxDelims: 0

```

On running the commands

```

cmh:~$ latexindent.pl tabular2.tex
cmh:~$ latexindent.pl -s tabular2.tex -l tabular2.yaml
cmh:~$ latexindent.pl -s tabular2.tex -l tabular3.yaml

```

we obtain the respective outputs given in Listings 32 to 34.

LISTING 32: tabular2-default.tex

```

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

```

LISTING 33: tabular2-mc.tex

```

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

```

LISTING 34: tabular2-no-max.tex

```

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

```

Notice in particular:

- in both Listings 32 and 33 all rows have been aligned at the ampersand, even those that do not contain the maximum number of ampersands (3 ampersands, in this case);



- in Listing 32 the columns have been aligned at the ampersand;
- in Listing 33 the `\multicolumn` command has grouped the 2 columns beneath *and* above it, because `multiColumnGrouping` is set to 1 in Listing 30;
- in Listing 34 rows 3 and 6 have *not* been aligned at the ampersand, because `alignRowsWithoutMaxDelims` has been set to 0 in Listing 31; however, the `\\` have still been aligned.

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 21); 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 matrix1.tex
```

then the before-and-after results shown in Listings 35 and 36 are achievable by default.

LISTING 35: matrix1.tex

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

LISTING 36: 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 37; the default output is shown in Listing 38. 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 37: align-block.tex

```
%*\begin{tabular}
uuu1_&2_&3_&4_\\
uuu5_&uuu6_&uuu\\
uu%*\end{tabular}
```

LISTING 38: align-block.tex default output

```
%*\begin{tabular}
#1_&2_&3_&4_\\
#5_&uuu6_&uuu\\
%*\end{tabular}
```

With reference to Table 2 on page 23 and the, yet undiscussed, fields of `noAdditionalIndent` and `indentRules` (see Section 5.2 on page 24), 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 40 and 41

LISTING 39: indentAfterItems

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



LISTING 40: 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 41: items1.tex default output

```
\begin{itemize}
  \item some_text_here
  \item some_more_text_here
  \item some_more_text_here
  \item another_item
  \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 11 for details of how to configure user settings, and Listing 9 on page 12 in particular.)

LISTING 42:
itemNames

```
itemNames:
  item: 1
  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 43 shows the default settings of `specialBeginEnd`.

LISTING 43: specialBeginEnd

```
190 specialBeginEnd:
191   displayMath:
192     begin: '\\\[
193     end: '\\\]'
194     lookForThis: 1
195   inlineMath:
196     begin: '(?!\\$)(?!\\$)\\$(!\\$)'
197     end: '(?!\\$)\\$(!\\$)'
198     lookForThis: 1
199   displayMathTeX:
200     begin: '\\$\\$'
201     end: '\\$\\$'
202     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 12); indeed, you might like to set up your own special begin and end statements.

A demonstration of the before-and-after results are shown in Listings 44 and 45.



LISTING 44: special1.tex before

```
The_function_ $ f $ _has_formula
\[
f(x)=x^2.
\]
If_you_like_splitting_dollars,
$
g(x)=f(2x)
$
```

LISTING 45: special1.tex default output

```
The_function_ $ f $ _has_formula
\[
    f(x)=x^2.
\]
If_you_like_splitting_dollars,
$
    g(x)=f(2x)
$
```

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

`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 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 page 24); 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 the code in Listing 47 saved into `headings1.yaml`, and that you have the text from Listing 48 saved into `headings1.tex`.

LISTING 47: headings1.yaml

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

LISTING 46: indentAfterHeadings

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

LISTING 48: 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

⁵There is a slight difference in interface for this field when comparing Version 2.2 to Version 3.0; see appendix C on page 67 for details.



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

then you should receive the output given in Listing 49.

LISTING 49: headings1.tex using Listing 47

```
\subsection{subsection_title}
  \subsection_text
  \subsection_text
  \paragraph{paragraph_title}
  \paragraph_text
  \paragraph_text
  \paragraph{paragraph_title}
  \paragraph_text
  \paragraph_text
```

LISTING 50: 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 47 so that the paragraph level is 1; after running

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

you should receive the code given in Listing 50; 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_\\	<code>\begin{myenv}</code> body of myenv <code>\end{myenv}</code>
optionalArguments	<i>inherits</i> name from parent (e.g environment name)	<code>[</code> opt arg text <code>]</code>
mandatoryArguments	<i>inherits</i> name from parent (e.g environment name)	<code>{</code> mand arg text <code>}</code>
commands	+a-zA-Z@*0-9_:	<code>\mycommand⟨arguments⟩</code>
keyEqualsValuesBracesBrackets	a-zA-Z@*0-9_/.\\h{\\}:\\#-	<code>my key/.style=⟨arguments⟩</code>
namedGroupingBracesBrackets	a-zA-Z@*⟩<	<code>in⟨arguments⟩</code>
UnNamedGroupingBracesBrackets	<i>No name!</i>	<code>{</code> or <code>[</code> or <code>,</code> or <code>&</code> or <code>)</code> or <code>(</code> or <code>\$</code> followed by <code>⟨arguments⟩</code>



ifElseFi	@a-zA-Z but must begin with either <code>\if</code> of <code>\@if</code>	<code>\ifnum...</code> ... <code>\else</code> ... <code>\fi</code>
----------	---	--

items	User specified, see Listings 39 and 42 on page 20 and on page 21	<code>\begin{enumerate}</code> <code>\item ...</code> <code>\end{enumerate}</code>
-------	---	--

specialBeginEnd	User specified, see Listing 43 on page 21	<code>\[</code> ... <code>\]</code>
-----------------	---	---

afterHeading	User specified, see Listing 46 on page 22	<code>\chapter{title}</code> ... <code>\section{title}</code>
--------------	---	---

filecontents	User specified, see Listing 18 on page 16...	<code>\begin{filecontents}</code> ... <code>\end{filecontents}</code>
--------------	--	---

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 23; for each type of code block in Table 2 on page 23 (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 23; for reference, there follows a list of the code blocks covered.

- 5.2.1 Environments and their arguments 25
- 5.2.2 Environments with items 31



5.2.3	Commands with arguments	32
5.2.4	ifelsefi code blocks	34
5.2.5	specialBeginEnd code blocks	36
5.2.6	afterHeading code blocks	37
5.2.7	The remaining code blocks	39
5.2.8	Summary	40

5.2.1 Environments and their arguments

There are a few different YAML switches governing the indentation of environments; let's start with the code shown in Listing 59.

LISTING 59: myenv.tex

```
\begin{outer}
\begin{myenv}
  \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 60 and 61.

LISTING 60:

myenv-noAdd1.yaml

```
noAdditionalIndent:
  myenv: 1
```

LISTING 61:

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

LISTING 62: myenv.tex output (using either Listing 60 or Listing 61)

```
\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 63 and 64, 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 65.

```
LISTING 63:
myenv-noAdd3.yaml

noAdditionalIndent:
  myenv: 0
```

```
LISTING 64:
myenv-noAdd4.yaml

noAdditionalIndent:
  myenv:
    body: 0
```

LISTING 65: myenv.tex output (using either Listing 63 or Listing 64)

```
\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 66.

LISTING 66: 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 67; 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 60), then *all* parts of the environment (body, optional and mandatory arguments) are assumed to want no additional indent.

LISTING 67: myenv-args.tex using Listing 60

```
\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 68 and 69.



LISTING 68: myenv-noAdd5.yaml

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

LISTING 69: 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 70 and 71. Note that in Listing 70 the text for the *optional* argument has not received any additional indentation, and that in Listing 71 the *mandatory* argument has not received any additional indentation; in both cases, the *body* has not received any additional indentation.

LISTING 70: myenv-args.tex using Listing 68

```
\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 71: myenv-args.tex using Listing 69

```
\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 72 and 73.

LISTING 72: myenv-rules1.yaml

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

LISTING 73: 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 74; note in particular that the environment `myenv` has received one tab (from the outer environment) plus three spaces from Listing 72 or 73.



LISTING 74: myenv.tex output (using either Listing 72 or Listing 73)

```

\begin{outer}
  \begin{myenv}
    \bodyofenvironment
    \bodyofenvironment
    \bodyofenvironment
  \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 66 that contains optional and mandatory arguments. Upon using Listing 72 as in

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

we obtain the output in Listing 75; note that the body, optional argument and mandatory argument have *all* received the same customised indentation.

LISTING 75: myenv-args.tex using Listing 72

```

\begin{outer}
  \begin{myenv}[%
    \optionalargumenttext
    \optionalargumenttext]%
    \{ \mandatoryargumenttext
    \mandatoryargumenttext}
    \bodyofenvironment
    \bodyofenvironment
    \bodyofenvironment
  \end{myenv}
\end{outer}

```

You can specify different indentation rules for the different features using, for example, Listings 76 and 77

LISTING 76: myenv-rules3.yaml

```

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

```

LISTING 77: 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 78 and 79.



LISTING 78: myenv-args.tex using Listing 76

```
\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 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}
```

Note that in Listing 78, 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 79, the optional argument has received the default (tab) indentation, the mandatory argument has received two tabs of indentation, and the body has received three spaces of indentation.

`noAdditionalIndentGlobal: {fields}`

Assuming that your environment name is not found within neither `noAdditionalIndent` nor `indentRules`, the next place that `latexindent.pl` will look is `noAdditionalIndentGlobal`, and in particular for the `environments` key (see Listing 80). Let's say that you change the value of `environments` to 1 in Listing 80, and that you run

LISTING 80:
`noAdditionalIndentGlobal`

```
noAdditionalIndentGlobal:
  environments: 0
```

```
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 81 and 82; in Listing 81 notice that *both* environments receive no additional indentation but that the arguments of `myenv` still *do* receive indentation. In Listing 82 notice that the `outer` environment does not receive additional indentation, but because of the settings from `myenv-rules1.yaml` (in Listing 72 on page 27), the `myenv` environment still *does* receive indentation.

LISTING 81: myenv-args.tex using Listing 80

```
\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 82: myenv-args.tex using Listings 72 and 80

```
\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 83 and 84



LISTING 83:
opt-args-no-add-glob.yaml

```
noAdditionalIndentGlobal:
  optionalArguments: 1
```

LISTING 84:
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 85 and 86. Notice that in Listing 85 the *optional* argument has not received any additional indentation, and in Listing 86 the *mandatory* argument has not received any additional indentation.

LISTING 85: myenv-args.tex using
Listing 83

```
\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 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}
```

```
indentRulesGlobal: {fields}
```

The final check that `latexindent.pl` will make is to look for `indentRulesGlobal` as detailed in Listing 87; if you change the `environments` field to anything involving horizontal space, say " ", and then run the following commands

LISTING 87:
indentRulesGlobal

```
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 88 and 89. Note that in Listing 88, both the environment blocks have received a single-space indentation, whereas in Listing 89 the outer environment has received single-space indentation (specified by `indentRulesGlobal`), but `myenv` has received " ", as specified by the particular `indentRules` for `myenv` Listing 72 on page 27.



LISTING 88: myenv-args.tex using Listing 87

```

\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 89: myenv-args.tex using Listings 72 and 87

```

\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 90 and 91

LISTING 90:
opt-args-indent-rules-glob.yaml

```

indentRulesGlobal:
  optionalArguments: "\t\t"

```

LISTING 91:
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 92 and 93. Note that the *optional* argument in Listing 92 has received two tabs worth of indentation, while the *mandatory* argument has done so in Listing 93.

LISTING 92: myenv-args.tex using Listing 90

```

\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 93: myenv-args.tex using Listing 91

```

\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 39 and 42 on page 20 and on page 21, some commands may contain item commands; for the purposes of this discussion, we will use the code from Listing 40 on page 21.

Assuming that you've populated `itemNames` with the name of your item, you can put the item name into `noAdditionalIndent` as in Listing 94, 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 95



LISTING 94: item-noAdd1.yaml

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

LISTING 95: 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 96 and 97; note that in Listing 96 that the text after each item has not received any additional indentation, and in Listing 97, the text after each item has received a single space of indentation, specified by Listing 95.

LISTING 96: items1.tex using Listing 94

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

LISTING 97: items1.tex using Listing 95

```
\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 98 and 99. Note that there is a need to ‘reset/remove’ the `item` field from `indentRules` in both cases (see the hierarchy description given on page 24) as the `item` command is a member of `indentRules` by default.

LISTING 98:

```
items-noAdditionalGlobal.yaml
```

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

LISTING 99:

```
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 96 and 97 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 100; when `latexindent.pl` operates on this file, the default output is shown in Listing 101.⁶

⁶The command code blocks have quite a few subtleties, described in Section 5.3 on page 41.



LISTING 100: mycommand.tex

```
\mycommand
{
mand_arg_text
mand_arg_text}
[
opt_arg_text
opt_arg_text
]
```

LISTING 101: 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 60 and 61 on page 25) we may specify `noAdditionalIndent` either in ‘scalar’ form, or in ‘field’ form, as shown in Listings 102 and 103

LISTING 102:

mycommand-noAdd1.yaml

```
noAdditionalIndent:
  mycommand: 1
```

LISTING 103:

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 104 and 105

LISTING 104: mycommand.tex using Listing 102

```
\mycommand
{
mand_arg_text
mand_arg_text}
[
opt_arg_text
opt_arg_text
]
```

LISTING 105: mycommand.tex using Listing 103

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}
[
  \opt_arg_text
  \opt_arg_text
]
```

Note that in Listing 104 that the ‘body’, optional argument *and* mandatory argument have *all* received no additional indentation, while in Listing 105, 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 68 and 69 on page 27; explicit examples are given in Listings 106 and 107.

LISTING 106:

mycommand-noAdd3.yaml

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

LISTING 107:

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 108 and 109.

LISTING 108: mycommand.tex using Listing 106

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}
[
opt_arg_text
opt_arg_text
]
```

LISTING 109: mycommand.tex using Listing 107

```
\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 108 and 109 has received no additional indent, even though body is explicitly set to 0 in both Listings 106 and 107. This is because, by default, noAdditionalIndentGlobal for commands is set to 1 by default; this can be easily fixed as in Listings 110 and 111.

LISTING 110:
mycommand-noAdd5.yaml

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

LISTING 111:
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 112 and 113.

LISTING 112: mycommand.tex using Listing 110

```
\mycommand
\{
\  \mand_arg_text
\  \mand_arg_text}
\[
\opt_arg_text
\opt_arg_text
\]
```

LISTING 113: mycommand.tex using Listing 111

```
\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 76 and 77 on page 28 and Listings 87, 90 and 91 on page 30 and on page 31.

5.2.4 ifelsefi code blocks

Let's use the simple example shown in Listing 114; when latexindent.pl operates on this file, the output as in Listing 115; note that the body of each of the \if statements have been indented, and that the \else statement has been accounted for correctly.



LISTING 114: ifelsefi1.tex	LISTING 115: ifelsefi1.tex default output
<pre> \ifodd\radius \ifnum\radius<14 \pgfmathparse{100-(\radius)*4}; \else \pgfmathparse{200-(\radius)*3}; \fi\fi </pre>	<pre> \ifodd\radius \ifnum\radius<14 \pgfmathparse{100-(\radius)*4}; \else \pgfmathparse{200-(\radius)*3}; \fi\fi </pre>

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 116 and 117.

LISTING 116: ifnum-noAdd.yaml	LISTING 117: ifnum-indent-rules.yaml
<pre> noAdditionalIndent: ifnum: 1 </pre>	<pre> indentRules: ifnum: " " </pre>

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 118 and 119; note that in Listing 118, the `ifnum` code block has not received any additional indentation, while in Listing 119, the `ifnum` code block has received one tab and two spaces of indentation.

LISTING 118: ifelsefi1.tex using Listing 116	LISTING 119: ifelsefi1.tex using Listing 117
<pre> \ifodd\radius \ifnum\radius<14 \pgfmathparse{100-(\radius)*4}; \else \pgfmathparse{200-(\radius)*3}; \fi\fi </pre>	<pre> \ifodd\radius \ifnum\radius<14 \pgfmathparse{100-(\radius)*4}; \else \pgfmathparse{200-(\radius)*3}; \fi\fi </pre>

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 120 and 121.

LISTING 120: ifelsefi-noAdd-glob.yaml	LISTING 121: ifelsefi-indent-rules-global.yaml
<pre> noAdditionalIndentGlobal: ifElseFi: 1 </pre>	<pre> indentRulesGlobal: ifElseFi: " " </pre>

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 122 and 123; notice that in Listing 122 neither of the `ifelsefi` code blocks have received indentation, while in Listing 123 both code blocks have received a single space of indentation.



LISTING 122: ifelsefi1.tex using Listing 120

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

LISTING 123: ifelsefi1.tex using Listing 121

```
\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 44 on page 22 which has default output shown in Listing 45 on page 22.

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 124 and 125.

LISTING 124:
displayMath-noAdd.yaml

```
noAdditionalIndent:
  displayMath: 1
```

LISTING 125:
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 126 and 127; note that in Listing 126, the `displayMath` code block has *not* received any additional indentation, while in Listing 127, the `displayMath` code block has received three tabs worth of indentation.

LISTING 126: special1.tex using Listing 124

```
The_\function_\$ f \$ _has_\formula
\[
f(x)=x^2.
\]
If_you_like_splitting_dollars,
$
  g(x)=f(2x)
$
```

LISTING 127: special1.tex using Listing 125

```
The_\function_\$ f \$ _has_\formula
\[
\t\t\tg(x)=x^2.
\]
If_you_like_splitting_dollars,
$
\t\t\tg(x)=f(2x)
$
```

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 128 and 129.

LISTING 128:
special-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
  specialBeginEnd: 1
```

LISTING 129:
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 130 and 131; notice that in Listing 130 neither of the special code blocks have received indentation, while in Listing 131 both code blocks have received a single space of indentation.

LISTING 130: special1.tex using Listing 128

```
The_function_ $ f $ _has_formula
\[
f(x)=x^2.
\]
If_you_like_splitting_dollars,
$
g(x)=f(2x)
$
```

LISTING 131: special1.tex using Listing 129

```
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 132 for demonstration throughout this Section. As discussed on page 22, by default latexindent.pl will not add indentation after headings.

LISTING 132: headings2.tex

```
\paragraph{paragraph
title}
paragraph_text
paragraph_text
```

On using the YAML file in Listing 134 by running the command

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

we obtain the output in Listing 133. Note that the argument of paragraph has received (default) indentation, and that the body after the heading statement has received (default) indentation.

LISTING 133: headings2.tex using Listing 134

```
\paragraph{paragraph
  ¶ title}
  ¶paragraph_text
  ¶paragraph_text
```

LISTING 134: headings3.yaml

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

If we specify noAdditionalIndent as in Listing 136 and run the command

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

then we receive the output in Listing 135. 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 135: headings2.tex using Listing 136

```
\paragraph{paragraph
title}
paragraph_text
paragraph_text
```

LISTING 136: headings4.yaml

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



Similarly, if we specify `indentRules` as in Listing 138 and run analogous commands to those above, we receive the output in Listing 137; note that the *body*, *mandatory argument* and *content after the heading* of paragraph have all received three tabs worth of indentation.

LISTING 137: headings2.tex using Listing 138

```
\paragraph{paragraph
  ¶   ¶   ¶   ¶   ¶   ¶   ¶   ¶   ¶   ¶title}
  ¶   ¶   ¶paragraph_text
  ¶   ¶   ¶paragraph_text
```

LISTING 138: headings5.yaml

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

We may, instead, specify `noAdditionalIndent` in ‘field’ form, as in Listing 140 which gives the output in Listing 139.

LISTING 139: headings2.tex using
Listing 140

```
\paragraph{paragraph
  ¶title}
paragraph_text
paragraph_text
```

LISTING 140: headings6.yaml

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

Analogously, we may specify `indentRules` as in Listing 142 which gives the output in Listing 141; 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 141: headings2.tex using
Listing 142

```
\paragraph{paragraph
  ¶    ¶    ¶_title}
  ¶    ¶    ¶paragraph_text
  ¶    ¶    ¶paragraph_text
```

LISTING 142: 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 144 and 146 respectively, with respective output in Listings 143 and 145. Note that in Listing 144 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 145, the *argument* has received both a (default) tab plus two spaces of indentation (from the global rule specified in Listing 146), and the remaining body after `paragraph` has received just two spaces of indentation.

LISTING 143: headings2.tex using
Listing 144

```
\paragraph{paragraph
  ¶title}
paragraph_text
paragraph_text
```

LISTING 144: headings8.yaml

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



LISTING 145: headings2.tex using
Listing 146

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

LISTING 146: 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 23, 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 32, 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 23;
- 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 147, with the default output given in Listing 148.

LISTING 147: pgfkeys1.tex

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

LISTING 148: pgfkeys1.tex default output

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

In Listing 148, 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 24.

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

- it must immediately follow either *horizontal space* OR *one or more line breaks* OR `{` OR `[` OR `$` OR `)` OR `;`;
- the name may contain the characters detailed in Table 2 on page 23;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

A simple example is given in Listing 149, with default output in Listing 150.

LISTING 149: child1.tex

```
\coordinate
child[grow=down]{
edge_\from_\parent_\[antiparticle]
node_\[above=3pt]_\{ $ C $ }
}
```

LISTING 150: 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`⁷. Referencing Listing 150, 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 24.

`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 151 with default output give in Listing 152.

LISTING 151: `psforeach1.tex`

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

LISTING 152: `psforeach1.tex` default output

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

Referencing Listing 152, 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 24.

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 153 and 154 should now make sense.

⁷ You may like to verify this by using the `-tt` option and checking `indent.log`!



LISTING 153: noAdditionalIndentGlobal

```

267 noAdditionalIndentGlobal:
268   environments: 0
269   commands: 1
270   optionalArguments: 0
271   mandatoryArguments: 0
272   ifElseFi: 0
273   items: 0
274   keyEqualsValuesBracesBrackets: 0
275   namedGroupingBracesBrackets: 0
276   UnNamedGroupingBracesBrackets: 0
277   specialBeginEnd: 0
278   afterHeading: 0
279   filecontents: 0

```

LISTING 154: indentRulesGlobal

```

283 indentRulesGlobal:
284   environments: 0
285   commands: 0
286   optionalArguments: 0
287   mandatoryArguments: 0
288   ifElseFi: 0
289   items: 0
290   keyEqualsValuesBracesBrackets: 0
291   namedGroupingBracesBrackets: 0
292   UnNamedGroupingBracesBrackets: 0
293   specialBeginEnd: 0
294   afterHeading: 0
295   filecontents: 0

```

5.3 Commands and the strings between their arguments

The command code blocks will always look for optional (square bracketed) and mandatory (curly braced) arguments which can contain comments, line breaks and ‘beamer’ commands `<.*?>` between them. There are switches that can allow them to contain other strings, which we discuss next.

```
commandCodeBlocks: {fields}
```

The `commandCodeBlocks` field contains a few switches detailed in Listing 155.

LISTING 155: commandCodeBlocks

```

298 commandCodeBlocks:
299   roundParenthesesAllowed: 1
300   stringsAllowedBetweenArguments:
301     node: 1
302     at: 1
303     to: 1
304     decoration: 1
305     ++: 1
306     --: 1

```

```
roundParenthesesAllowed: 0|1
```

The need for this field was mostly motivated by commands found in code used to generate images in PSTricks and tikz; for example, let’s consider the code given in Listing 156.

LISTING 156: pstricks1.tex

```

\defFunction[algebraic]{torus}(u,v)
{(2+cos(u))*cos(v+\Pi)}
{(2+cos(u))*sin(v+\Pi)}
{sin(u)}

```

LISTING 157: pstricks1 default output

```

\defFunction[algebraic]{torus}(u,v)
{(2+cos(u))*cos(v+\Pi)}
{(2+cos(u))*sin(v+\Pi)}
{sin(u)}

```

Notice that the `\defFunction` command has an optional argument, followed by a mandatory argument, followed by a round-parenthesis argument, (u, v) .

By default, because `roundParenthesesAllowed` is set to 1 in Listing 155, then `latexindent.pl` will allow round parenthesis between optional and mandatory arguments. In the case of the code in Listing 156, `latexindent.pl` finds *all* the arguments of `defFunction`, both before and after (u, v) .

The default output from running `latexindent.pl` on Listing 156 actually leaves it unchanged (see Listing 157); note in particular, this is because of `noAdditionalIndentGlobal` as discussed on page 34.

Upon using the YAML settings in Listing 159, and running the command



```
cmh:~$ latexindent.pl pstricks1.tex -l noRoundParentheses.yaml
```

we obtain the output given in Listing 158.

LISTING 158: pstricks1.tex using Listing 159

```
\defFunction[algebraic]{torus}(u,v)
{(2+cos(u))*cos(v+\Pi)}
*{(2+cos(u))*sin(v+\Pi)}
*{sin(u)}
```

LISTING 159:

noRoundParentheses.yaml

```
commandCodeBlocks:
  roundParenthesesAllowed: 0
```

Notice the difference between Listing 157 and Listing 158; in particular, in Listing 158, because round parentheses are *not* allowed, latexindent.pl finds that the `\defFunction` command finishes at the first opening round parenthesis. As such, the remaining braced, mandatory, arguments are found to be `UnNamedGroupingBracesBrackets` (see Table 2 on page 23) which, by default, assume indentation for their body, and hence the tabbed indentation in Listing 158.

Let's explore this using the YAML given in Listing 161 and run the command

```
cmh:~$ latexindent.pl pstricks1.tex -l defFunction.yaml
```

then the output is as in Listing 160.

LISTING 160: pstricks1.tex using Listing 161

```
\defFunction[algebraic]{torus}(u,v)
└{(2+cos(u))*cos(v+\Pi)}
└{(2+cos(u))*sin(v+\Pi)}
└{sin(u)}
```

LISTING 161: defFunction.yaml

```
indentRules:
  defFunction:
    body: " "
```

Notice in Listing 160 that the *body* of the `defFunction` command i.e, the subsequent lines containing arguments after the command name, have received the single space of indentation specified by Listing 161.

`stringsAllowedBetweenArguments:` `<fields>`

tikz users may well specify code such as that given in Listing 162; processing this code using latexindent.pl gives the default output in Listing 163.

LISTING 162: tikz-node1.tex

```
\draw[thin]
(c)└to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]
```

LISTING 163: tikz-node1 default output

```
\draw[thin]
(c)└to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]
```

With reference to Listing 155 on page 41, we see that the strings

to, node, ++

are all allowed to appear between arguments, as they are each set to 1; importantly, you are encouraged to add further names to this field as necessary. This means that when latexindent.pl processes Listing 162, it consumes:

- the optional argument `[thin]`
- the round-bracketed argument `(c)` because `roundParenthesesAllowed` is 1 by default
- the string `to` (specified in `stringsAllowedBetweenArguments`)



- the optional argument [in=110,out=-90]
- the string ++ (specified in stringsAllowedBetweenArguments)
- the round-bracketed argument (0,-0.5cm) because roundParenthesesAllowed is 1 by default
- the string node (specified in stringsAllowedBetweenArguments)
- the optional argument [below,align=left,scale=0.5]

We can explore this further, for example using Listing 165 and running the command

```
cmh:~$ latexindent.pl tikz-node1.tex -l draw.yaml
```

we receive the output given in Listing 164.

LISTING 164: tikz-node1.tex using Listing 165

```
\draw[thin]
  (c) to[in=110,out=-90]
  ++(0,-0.5cm)
  node[below,align=left,scale=0.5]
```

LISTING 165: draw.yaml

```
indentRules:
  draw:
    body: " "
```

Notice that each line after the \draw command (its ‘body’) in Listing 164 has been given the appropriate two-spaces worth of indentation specified in Listing 165.

Let’s compare this with the output from using the YAML settings in Listing 167, and running the command

```
cmh:~$ latexindent.pl tikz-node1.tex -l no-to.yaml
```

given in Listing 166.

LISTING 166: tikz-node1.tex using Listing 167

```
\draw[thin]
(c) to[in=110,out=-90]
++(0,-0.5cm)
node[below,align=left,scale=0.5]
```

LISTING 167: no-to.yaml

```
commandCodeBlocks:
  stringsAllowedBetweenArguments:
    to: 0
```

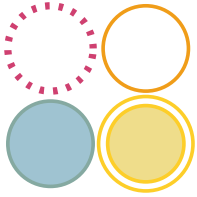
In this case, latexindent.pl sees that:

- the \draw command finishes after the (c) as (stringsAllowedBetweenArguments has to set to 0)
- it finds a namedGroupingBracesBrackets called to (see Table 2 on page 23) with argument [in=110,out=-90]
- it finds another namedGroupingBracesBrackets but this time called node with argument [below,align=left,scale=0.5]

6 The -m (modifylinebreaks) switch

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

```
modifylinebreaks: <fields>
```



As of Version 3.0, `latexindent.pl` has the `-m` switch, which permits `latexindent.pl` to modify line breaks, according to the specifications in the `modifyLineBreaks` field. *The settings in this field will only be considered if the `-m` switch has been used.* A snippet of the default settings of this field is shown in Listing 168.

LISTING 168: `modifyLineBreaks`

```

376 modifyLineBreaks:
377     preserveBlankLines: 1
378     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 169 shows a sample file with blank lines; upon running

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

the output is shown in Listing 170; note that the multiple blank lines have been condensed into one blank line, and note also that we have used the `-m` switch!

LISTING 169: `mlb1.tex`

before_␣blank_␣line

after_␣blank_␣line

after_␣blank_␣line

LISTING 170: `mlb1.tex` out output

before_␣blank_␣line

after_␣blank_␣line

after_␣blank_␣line

`textWrapOptions: <fields>`

N: 2017-05-27

When the `-m` switch is active `latexindent.pl` has the ability to wrap text using the options specified in the `textWrapOptions` field, see Listing 171. The value of `columns` specifies the column at which the text should be wrapped. By default, the value of `columns` is 0, so `latexindent.pl` will *not* wrap text; if you change it to a value of 2 or more, then text will be wrapped after the character in the specified column.



LISTING 171: textWrapOptions

```
379 textWrapOptions:
380 columns: 0
```

For example, consider the file give in Listing 172.

LISTING 172: textwrap1.tex

```
Here is a line of text that will be wrapped by latexindent.pl. Each line is quite long.
Here is a line of text that will be wrapped by latexindent.pl. Each line is quite long.
```

Using the file textwrap1.yaml in Listing 174, and running the command

```
cmh:~$ latexindent.pl -m textwrap1.tex -o textwrap1-mod1.tex -l textwrap1.yaml
```

we obtain the output in Listing 173.

LISTING 173: textwrap1-mod1.tex

```
Here is a line of
text that will be
wrapped by
latexindent.pl.
Each line is quite
long.

Here is a line of
text that will be
wrapped by
latexindent.pl.
Each line is quite
long.
```

LISTING 174: textwrap1.yaml

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

The text wrapping routine is performed *after* verbatim environments have been stored, so verbatim environments and verbatim commands are exempt from the routine. For example, using the file in Listing 175,

LISTING 175: textwrap2.tex

```
Here is a line of text that will be wrapped by latexindent.pl. Each line is quite long.

\begin{verbatim}
    a long line in a verbatim environment, which will not be broken by latexindent.pl
\end{verbatim}

Here is a verb command: \verb!this will not be text wrapped!
```

and running the following command and continuing to use textwrap1.yaml from Listing 174,

```
cmh:~$ latexindent.pl -m textwrap2.tex -o textwrap2-mod1.tex -l textwrap1.yaml
```

then the output is as in Listing 176.



LISTING 176: textwrap2-mod1.tex

```
Here_is_a_line_of
text_that_will_be
wrapped_by
latexindent.pl.
Each_line_is_quite
long.

\begin{verbatim}
      a long line in a verbatim environment, which will not be broken by latexindent.pl
\end{verbatim}

Here_is_a_verb
command:
\verb!this_will_not_be_text_wrapped!
```

Furthermore, the text wrapping routine is performed after the trailing comments have been stored, and they are also exempt from text wrapping. For example, using the file in Listing 177

LISTING 177: textwrap3.tex

```
Here_is_a_line_of_text_that_will_be_wrapped_by_latexindent.pl. Each_line_is_quite_long.

Here_is_a_line_%_text_wrapping_does_not_apply_to_comments_by_latexindent.pl
```

and running the following command and continuing to use textwrap1.yaml from Listing 174,

```
cmh:~$ latexindent.pl -m textwrap3.tex -o textwrap3-mod1.tex -l textwrap1.yaml
```

then the output is as in Listing 178.

LISTING 178: textwrap3-mod1.tex

```
Here_is_a_line_of
text_that_will_be
wrapped_by
latexindent.pl.
Each_line_is_quite
long.

Here_is_a_line
%_text_wrapping_does_not_apply_to_comments_by_latexindent.pl
```

The text wrapping routine of latexindent.pl is performed by the Text::Wrap module, which provides a separator feature to separate lines with characters other than a new line (see [11]). By default, the separator is empty (see Listing 179) which means that a new line token will be used, but you can change it as you see fit.

LISTING 179: textWrapOptions

```
textWrapOptions:
  columns: 0
  separator: ""
```

For example starting with the file in Listing 180

LISTING 180: textwrap4.tex

```
Here_is_a_line_of_text.
```

and using textwrap2.yaml from Listing 182 with the following command



```
cmh:~$ latexindent.pl -m textwrap4.tex -o textwrap4-mod2.tex -l textwrap2.yaml
```

then we obtain the output in Listing 181.

LISTING 181: textwrap4-mod2.tex

```
Here||is_a||line||of||text||.
```

LISTING 182: textwrap2.yaml

-m

```
modifyLineBreaks:
  textWrapOptions:
    columns: 5
    separator: "||"
```

Summary of text wrapping It is important to note the following:

- Verbatim environments (Listing 13 on page 15) and verbatim commands (Listing 14 on page 15) will *not* be affected by the text wrapping routine (see Listing 176 on page 46);
- comments will *not* be affected by the text wrapping routine (see Listing 178 on page 46);
- indentation is performed *after* the text wrapping routine; as such, indented code will likely exceed any maximum value set in the `columns` field.

`removeParagraphLineBreaks`: *<fields>*

N: 2017-05-27

When the `-m` switch is active `latexindent.pl` has the ability to remove line breaks from within paragraphs; the behaviour is controlled by the `removeParagraphLineBreaks` field, detailed in Listing 183. Thank you to [8] for shaping and assisting with the testing of this feature.

LISTING 183: `removeParagraphLineBreaks`

-m

```
382   removeParagraphLineBreaks:
383     all: 0
384     alignAtAmpersandTakesPriority: 1
385     environments:
386       quotation: 0
387       ifElseFi: 0
388     optionalArguments: 0
389     mandatoryArguments: 0
390     items: 0
391     specialBeginEnd: 0
392     afterHeading: 0
393     filecontents: 0
394     masterDocument: 0
```

This routine can be turned on *globally* for *every* code block type known to `latexindent.pl` (see Table 2 on page 23) by using the `all` switch; by default, this switch is *off*. Assuming that the `all` switch is off, then the routine can be controlled on a per-code-block-type basis, and within that, on a per-name basis. We will consider examples of each of these in turn, but before we do, let's specify what `latexindent.pl` considers as a paragraph:

- it must begin on its own line with either an alphabetic or numeric character, and not with any of the code-block types detailed in Table 2 on page 23;
- it can include line breaks, but finishes when it meets either a blank line, a `\par` command, or any of the user-specified settings in the `paragraphsStopAt` field, detailed in Listing 200 on page 52.

Let's start with the `.tex` file in Listing 184, together with the YAML settings in Listing 185.



LISTING 184: shortlines.tex

```
\begin{myenv}
The_lines
in_this
environment
are_very
short
and_contain
many_linebreaks.
```

```
Another
paragraph.
\end{myenv}
```

Upon running the command

```
cmh:~$ latexindent.pl -m shortlines.tex -o shortlines1.tex -l remove-para1.yaml
```

then we obtain the output given in Listing 186.

LISTING 186: shortlines1.tex

```
\begin{myenv}
  ¶The_lines_in_this_environment_are_very_short_and_contain_many_linebreaks.

  ¶Another_paragraph.
\end{myenv}
```

Keen readers may notice that some trailing white space must be present in the file in Listing 184 which has crept in to the output in Listing 186. This can be fixed using the YAML file in Listing 239 on page 58 and running, for example,

```
cmh:~$ latexindent.pl -m shortlines.tex -o shortlines1-tws.tex -l
remove-para1.yaml,removeTWS-before.yaml
```

in which case the output is as in Listing 187; notice that the double spaces present in Listing 186 have been addressed.

LISTING 187: shortlines1-tws.tex

```
\begin{myenv}
  ¶The_lines_in_this_environment_are_very_short_and_contain_many_linebreaks.

  ¶Another_paragraph.
\end{myenv}
```

Keeping with the settings in Listing 185, we note that the `all` switch applies to *all* code block types. So, for example, let's consider the files in Listings 188 and 189

LISTING 185: remove-para1.yaml

-m

```
modifyLineBreaks:
  removeParagraphLineBreaks:
    all: 1
```



LISTING 188: shortlines-mand.tex

```
\mycommand{
The_lines
in_this
command
are_very
short
and_contain
many_linebreaks.

Another
paragraph.
}
```

LISTING 189: shortlines-opt.tex

```
\mycommand[
The_lines
in_this
command
are_very
short
and_contain
many_linebreaks.

Another
paragraph.
]
```

Upon running the commands

```
cmh:~$ latexindent.pl -m shortlines-mand.tex -o shortlines-mand1.tex -l remove-para1.yaml
cmh:~$ latexindent.pl -m shortlines-opt.tex -o shortlines-opt1.tex -l remove-para1.yaml
```

then we obtain the respective output given in Listings 190 and 191.

LISTING 190: shortlines-mand1.tex

```
\mycommand{
  ¶The_lines_in_this_command_are_very_short_and_contain_many_linebreaks.

  ¶Another_paragraph.
}
```

LISTING 191: shortlines-opt1.tex

```
\mycommand[
  ¶The_lines_in_this_command_are_very_short_and_contain_many_linebreaks.

  ¶Another_paragraph.
]
```

Assuming that we turn *off* the all switch (by setting it to 0), then we can control the behaviour of `removeParagraphLineBreaks` either on a per-code-block-type basis, or on a per-name basis.

For example, let's use the code in Listing 192, and consider the settings in Listings 193 and 194; note that in Listing 193 we specify that *every* environment should receive treatment from the routine, while in Listing 194 we specify that *only* the one environment should receive the treatment.



LISTING 192: shortlines-envs.tex

```

\begin{one}
The_lines
in_this
environment
are_very
short
and_contain
many_linebreaks.

Another
paragraph.
\end{one}

\begin{two}
The_lines
in_this
environment
are_very
short
and_contain
many_linebreaks.

Another
paragraph.
\end{two}

```

Upon running the commands

```

cmh:~$ latexindent.pl -m shortlines-envs.tex -o shortlines-envs2.tex -l remove-para2.yaml
cmh:~$ latexindent.pl -m shortlines-envs.tex -o shortlines-envs3.tex -l remove-para3.yaml

```

then we obtain the respective output given in Listings 195 and 196.

LISTING 195: shortlines-envs2.tex

```

\begin{one}
  ¶The_lines_in_this_environment_are_very_short_and_contain_many_linebreaks.

  ¶Another_paragraph.
\end{one}

\begin{two}
  ¶The_lines_in_this_environment_are_very_short_and_contain_many_linebreaks.

  ¶Another_paragraph.
\end{two}

```

LISTING 193: remove-para2.yaml

-m

```

modifyLineBreaks:
  removeParagraphLineBreaks:
    environments: 1

```

LISTING 194: remove-para3.yaml

-m

```

modifyLineBreaks:
  removeParagraphLineBreaks:
    environments:
      one: 1

```



LISTING 196: shortlines-envs3.tex

```

\begin{one}
  \The_lines_in_this_environment_are_very_short_and_contain_many_linebreaks.

  \Another_paragraph.
\end{one}

\begin{two}
  \The_lines
  \in_this
  \environment
  \are_very
  \short
  \and_contain
  \many_linebreaks.

  \Another
  \paragraph.
\end{two}

```

The remaining code-block types can be customized in analogous ways, although note that commands, `keyEqualsValuesBracesBrackets`, `namedGroupingBracesBrackets`, `UnNamedGroupingBracesBrackets` are controlled by the `optionalArguments` and the `mandatoryArguments`.

The only special case is the `masterDocument` field; this is designed for ‘chapter’-type files that may contain paragraphs that are not within any other code-blocks. For example, consider the file in Listing 197, with the YAML settings in Listing 198.

LISTING 197: shortlines-md.tex

```

The_lines
in_this
document
are_very
short
and_contain
many_linebreaks.

Another
paragraph.

\begin{myenv}
The_lines
in_this
document
are_very
short
and_contain
many_linebreaks.
\end{myenv}

```

LISTING 198: remove-para4.yaml

-m

```

modifyLineBreaks:
  removeParagraphLineBreaks:
    masterDocument: 1

```

Upon running the following command

```
cmh:~$ latexindent.pl -m shortlines-md.tex -o shortlines-md4.tex -l remove-para4.yaml
```

then we obtain the output in Listing 199.



LISTING 199: shortlines-md4.tex

```
The_lines_in_this_document_are_very_short_and_contain_many_linebreaks.
```

```
Another_paragraph.
```

```
\begin{myenv}
  %The_lines
  %in_this
  %document
  %are_very
  %short
  %and_contain
  %many_linebreaks.
\end{myenv}
```

`paragraphsStopAt: {fields}`

N: 2017-05-27

The paragraph line break routine considers blank lines and the `\par` command to be the end of a paragraph; you can fine tune the behaviour of the routine further by using the `paragraphsStopAt` fields, shown in Listing 200.

LISTING 200: paragraphsStopAt

```
395 paragraphsStopAt:
396   environments: 1
397   commands: 0
398   ifElseFi: 0
399   items: 0
400   specialBeginEnd: 0
401   heading: 0
402   filecontents: 0
403   comments: 0
```

The fields specified in `paragraphsStopAt` tell `latexindent.pl` to stop the current paragraph when it reaches a line that *begins* with any of the code-block types specified as 1 in Listing 200. By default, you'll see that the paragraph line break routine will stop when it reaches an environment at the beginning of a line. It is *not* possible to specify these fields on a per-name basis.

Let's use the `.tex` file in Listing 201; we will, in turn, consider the settings in Listings 202 and 203.

LISTING 201: sl-stop.tex

```
These_lines
are_very
short
\emph{and}_contain
many_linebreaks.
\begin{myenv}
Body_of_myenv
\end{myenv}
```

```
Another
paragraph.
%_a_comment
%_a_comment
```

LISTING 202: stop-command.yaml

```
modifyLineBreaks:
  removeParagraphLineBreaks:
    paragraphsStopAt:
      commands: 1
```

LISTING 203: stop-comment.yaml

```
modifyLineBreaks:
  removeParagraphLineBreaks:
    paragraphsStopAt:
      comments: 1
```

Upon using the settings from Listing 198 on page 51 and running the commands



```
cmh:~$ latexindent.pl -m sl-stop.tex -o sl-stop4.tex -l remove-para4.yaml
cmh:~$ latexindent.pl -m sl-stop.tex -o sl-stop4-command.tex -l=remove-para4.yaml,stop-command.yaml
cmh:~$ latexindent.pl -m sl-stop.tex -o sl-stop4-comment.tex -l=remove-para4.yaml,stop-comment.yaml
```

we obtain the respective outputs in Listings 204 to 206; notice in particular that:

- in Listing 204 the paragraph line break routine has included commands and comments;
- in Listing 205 the paragraph line break routine has *stopped* at the `\emph` command, because in Listing 202 we have specified commands to be 1, and `\emph` is at the beginning of a line;
- in Listing 206 the paragraph line break routine has *stopped* at the comments, because in Listing 203 we have specified comments to be 1, and the comment is at the beginning of a line.

In all outputs in Listings 204 to 206 we notice that the paragraph line break routine has stopped at `\begin{myenv}` because, by default, environments is set to 1 in Listing 200 on page 52.

LISTING 204: sl-stop4.tex

```
These lines are very short \emph{and} contain many linebreaks.
\begin{myenv}
  Body of myenv
\end{myenv}

Another paragraph. %a comment %a comment
```

LISTING 205: sl-stop4-command.tex

```
These lines are very short
\emph{and} contain
many linebreaks.
\begin{myenv}
  Body of myenv
\end{myenv}

Another paragraph. %a comment %a comment
```

LISTING 206: sl-stop4-comment.tex

```
These lines are very short \emph{and} contain many linebreaks.
\begin{myenv}
  Body of myenv
\end{myenv}

Another paragraph.
%a comment
%a comment
```

6.1 Poly-switches

Every other field in the `modifyLineBreaks` field uses poly-switches, and can take one of four integer values⁸:

- 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>*;

⁸You might like to associate one of the four circles in the logo with one of the four given values



- 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 207; note that it contains *global* settings (immediately after the `environments` field) and that *per-name* settings are also allowed – in the case of Listing 207, settings for `equation*` have been specified. Note that all poly-switches are *off* by default.

LISTING 207: environments -m

```

404 environments:
405     BeginStartsOnOwnLine: 0
406     BodyStartsOnOwnLine: 0
407     EndStartsOnOwnLine: 0
408     EndFinishesWithLineBreak: 0
409 equation*:
410     BeginStartsOnOwnLine: 0
411     BodyStartsOnOwnLine: 0
412     EndStartsOnOwnLine: 0
413     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 208; note that we have annotated key parts of the file using ♠, ♥, ♦ and ♣, these will be related to fields specified in Listing 207.

LISTING 208: env-mlb1.tex

```

before words ♠ \begin{myenv} ♥ body of myenv ♦ \end{myenv} ♣ after words

```

Let's explore `BeginStartsOnOwnLine` and `BodyStartsOnOwnLine` in Listings 209 and 210, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 209: env-mlb1.yaml -m

```

modifyLineBreaks:
  environments:
    BeginStartsOnOwnLine: 1

```

LISTING 210: 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 211 and 212 respectively.

LISTING 211: env-mlb.tex using Listing 209

```

before_words
\begin{myenv}body_of_myenv\end{myenv}after_words

```

LISTING 212: env-mlb.tex using Listing 210

```

before_words \begin{myenv}
      body_of_myenv\end{myenv}after_words

```

There are a couple of points to note:

- in Listing 211 a line break has been added at the point denoted by ♠ in Listing 208; no other line breaks have been changed;
- in Listing 212 a line break has been added at the point denoted by ♥ in Listing 208; furthermore, note that the *body* of `myenv` has received the appropriate (default) indentation.



Let's now change each of the 1 values in Listings 209 and 210 so that they are 2 and save them into env-mlb3.yaml and env-mlb4.yaml respectively (see Listings 213 and 214).

LISTING 213: env-mlb3.yaml -m

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

LISTING 214: env-mlb4.yaml -m

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

Upon running commands analogous to the above, we obtain Listings 215 and 216.

LISTING 215: env-mlb.tex using Listing 213

```
before_words%
\begin{myenv}body_of_myenv\end{myenv}after_words
```

LISTING 216: env-mlb.tex using Listing 214

```
before_words\begin{myenv}%
#body_of_myenv\end{myenv}after_words
```

Note that line breaks have been added as in Listings 211 and 212, 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 217 and 218, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 217: env-mlb5.yaml -m

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

LISTING 218: 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 219 and 220.

LISTING 219: env-mlb.tex using Listing 217

```
before_words\begin{myenv}body_of_myenv
\end{myenv}after_words
```

LISTING 220: env-mlb.tex using Listing 218

```
before_words\begin{myenv}body_of_myenv\end{myenv}
after_words
```

There are a couple of points to note:

- in Listing 219 a line break has been added at the point denoted by ♦ in Listing 208 on page 54; no other line breaks have been changed and the `\end{myenv}` statement has *not* received indentation (as intended);
- in Listing 220 a line break has been added at the point denoted by ♣ in Listing 208 on page 54.

Let's now change each of the 1 values in Listings 217 and 218 so that they are 2 and save them into env-mlb7.yaml and env-mlb8.yaml respectively (see Listings 221 and 222).

LISTING 221: env-mlb7.yaml -m

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

LISTING 222: env-mlb8.yaml -m

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

Upon running commands analogous to the above, we obtain Listings 223 and 224.

LISTING 223: env-mlb.tex using Listing 221

```
before_words\begin{myenv}body_of_myenv%
\end{myenv}after_words
```

LISTING 224: env-mlb.tex using Listing 222

```
before_words\begin{myenv}body_of_myenv\end{myenv}%
after_words
```

Note that line breaks have been added as in Listings 219 and 220, 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 225 using any of the YAML files presented so far in this section, it will be left unchanged.

LISTING 225: env-mlb2.tex	LISTING 226: env-mlb3.tex
<code>before_words</code> <code>\begin{myenv}</code> <code>body_of_myenv</code> <code>\end{myenv}</code> <code>after_words</code>	<code>before_words</code> <code>\begin{myenv}_%</code> <code>body_of_myenv%</code> <code>\end{myenv}_%</code> <code>after_words</code>

In contrast, the output from processing the file in Listing 226 will vary depending on the poly-switches used; in Listing 227 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 228 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 226 and by setting the other poly-switches considered so far to 2 in turn.

LISTING 227: env-mlb3.tex using Listing 210 on page 54	LISTING 228: env-mlb3.tex using Listing 214 on page 55
<code>before_words</code> <code>\begin{myenv}</code> <code> %</code> <code> body_of_myenv%</code> <code>\end{myenv}_%</code> <code>after_words</code>	<code>before_words</code> <code>\begin{myenv}_%</code> <code> body_of_myenv%</code> <code>\end{myenv}_%</code> <code>after_words</code>

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 207 on page 54, 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 229, noting in particular the positions of the line break highlighters, ♠, ♥, ♦ and ♣, together with the associated YAML files in Listings 230 to 233.



LISTING 229: env-mlb4.tex

```
before words♠
\begin{myenv}♥
body of myenv◇
\end{myenv}♣
after words
```

LISTING 230: env-mlb9.yaml

-m

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

LISTING 231: env-mlb10.yaml

-m

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

LISTING 232: env-mlb11.yaml

-m

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

LISTING 233: 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 234 to 237.

LISTING 234: env-mlb4.tex using Listing 230

```
before_words\begin{myenv}
  #body_of_myenv
\end{myenv}
after_words
```

LISTING 235: env-mlb4.tex using Listing 231

```
before_words
\begin{myenv}body_of_myenv
\end{myenv}
after_words
```

LISTING 236: env-mlb4.tex using Listing 232

```
before_words
\begin{myenv}
  #body_of_myenv\end{myenv}
after_words
```

LISTING 237: env-mlb4.tex using Listing 233

```
before_words
\begin{myenv}
  #body_of_myenv
\end{myenv}after_words
```

Notice that in

- Listing 234 the line break denoted by ♠ in Listing 229 has been removed;
- Listing 235 the line break denoted by ♥ in Listing 229 has been removed;
- Listing 236 the line break denoted by ◇ in Listing 229 has been removed;
- Listing 237 the line break denoted by ♣ in Listing 229 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 230 to 233 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 208 on page 54.

About trailing horizontal space Recall that on page 16 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 238, which highlights trailing spaces.

LISTING 238: `env-mlb5.tex`

```
before_words
\begin{myenv}
body of myenv
\end{myenv}
after_words
```

LISTING 239:

`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 240 and 241; note that the trailing horizontal white space has been preserved (by default) in Listing 240, while in Listing 241, it has been removed using the switch specified in Listing 239.

LISTING 240: `env-mlb5.tex` using Listings 234 to 237

```
before_words \begin{myenv} body of myenv \end{myenv} after_words
```

LISTING 241: `env-mlb5.tex` using Listings 234 to 237 and Listing 239

```
before_words \begin{myenv} body of myenv \end{myenv} after_words
```

Blank lines Now let's consider the file in Listing 242, which contains blank lines.

LISTING 242: `env-mlb6.tex`

```
before words

\begin{myenv}

body of myenv

\end{myenv}

after words
```

LISTING 243:

`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 244 and 245. In Listing 244 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 245, we have allowed the poly-switches to remove blank lines because, in Listing 243, we have set `preserveBlankLines` to 0.

LISTING 244: `env-mlb6.tex`
using Listings 234 to 237

before_words

`\begin{myenv}`

`\body_of_myenv`

`\end{myenv}`

after_words

LISTING 245: `env-mlb6.tex` using Listings 234 to 237 and Listing 243

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 54), 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 37 on page 20) 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♠ <code>\begin{myenv}</code> ♥ body of myenv◇ <code>\end{myenv}</code> ♣ after words	♠ <code>BeginStartsOnOwnLine</code> ♥ <code>BodyStartsOnOwnLine</code> ◇ <code>EndStartsOnOwnLine</code> ♣ <code>EndFinishesWithLineBreak</code>
ifelsefi	before words♠ <code>\if...</code> ♥ body of if statement★ <code>\else</code> □ body of else statement◇ <code>\fi</code> ♣ after words	♠ <code>IfStartsOnOwnLine</code> ♥ <code>BodyStartsOnOwnLine</code> ★ <code>ElseStartsOnOwnLine</code> □ <code>ElseFinishesWithLineBreak</code> ◇ <code>FiStartsOnOwnLine</code> ♣ <code>FiFinishesWithLineBreak</code>
optionalArguments	...♠ <code>[</code> ♥ body of opt arg◇ <code>]</code> ♣ ...	♠ <code>LSqBStartsOnOwnLine</code> ⁹ ♥ <code>OptArgBodyStartsOnOwnLine</code> ◇ <code>RSqBStartsOnOwnLine</code> ♣ <code>RSqBFinishesWithLineBreak</code>

⁹LSqB stands for Left Square Bracket



mandatoryArguments	<pre> ...♠ {♥ body of mand arg◇ }♣ ... </pre>	<p>♠ LCuBStartsOnOwnLine¹⁰</p> <p>♥ MandArgBodyStartsOnOwnLine</p> <p>◇ RCuBStartsOnOwnLine</p> <p>♣ RCuBFinishesWithLineBreak</p>
commands	<pre> before words♠ \mycommand♥ ⟨arguments⟩ </pre>	<p>♠ CommandStartsOnOwnLine</p> <p>♥ CommandNameFinishesWithLineBreak</p>
namedGroupingBraces Brackets	<pre> before words♠ myname♥ ⟨braces/brackets⟩ </pre>	<p>♠ NameStartsOnOwnLine</p> <p>♥ NameFinishesWithLineBreak</p>
keyEqualsValuesBraces Brackets	<pre> before words♠ key=♥ ⟨braces/brackets⟩ </pre>	<p>♠ KeyStartsOnOwnLine</p> <p>• EqualsStartsOnOwnLine</p> <p>♥ EqualsFinishesWithLineBreak</p>
items	<pre> before words♠ \item♥ ... </pre>	<p>♠ ItemStartsOnOwnLine</p> <p>♥ ItemFinishesWithLineBreak</p>
specialBeginEnd	<pre> before words♠ \[♥ body of special◇ \]♣ after words </pre>	<p>♠ SpecialBeginStartsOnOwnLine</p> <p>♥ SpecialBodyStartsOnOwnLine</p> <p>◇ SpecialEndStartsOnOwnLine</p> <p>♣ SpecialEndFinishesWithLineBreak</p>

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 59) and LCuBStartsOnOwnLine for mandatory arguments, and LSqBStartsOnOwnLine for optional arguments.

Let's begin with the code in Listing 255 and the YAML settings in Listing 257; with reference to Table 3 on page 59, the key CommandNameFinishesWithLineBreak is an alias for BodyStartsOnOwnLine.

¹⁰LCuB stands for Left Curly Brace



LISTING 255: 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 256; note that the *second* mandatory argument beginning brace { has had its leading line break removed, but that the *first* brace has not.

LISTING 256: mycommand1.tex
using Listing 257

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}{
  \mand_arg_text
  \mand_arg_text}
```

LISTING 257: mycom-mlb1.yaml

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

Now let's change the YAML file so that it is as in Listing 259; upon running the analogous command to that given above, we obtain Listing 258; both beginning braces { have had their leading line breaks removed.

LISTING 258: mycommand1.tex
using Listing 259

```
\mycommand{
  \mand_arg_text
  \mand_arg_text}{
  \mand_arg_text
  \mand_arg_text}
```

LISTING 259: mycom-mlb2.yaml

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

Now let's change the YAML file so that it is as in Listing 261; upon running the analogous command to that given above, we obtain Listing 260.

LISTING 260: mycommand1.tex
using Listing 261

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}
{
  \mand_arg_text
  \mand_arg_text}
```

LISTING 261: mycom-mlb3.yaml

```
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 255, and consider the YAML settings given in Listing 263. The output from running

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



is given in Listing 263.

LISTING 262: mycommand1.tex
using Listing 263

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}{
  \mand_arg_text
  \mand_arg_text}
```

LISTING 263: mycom-mlb4.yaml

-m

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

Studying Listing 263, 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 262, it is clear that LCuBStartsOnOwnLine won this conflict, and the reason is that *the second argument was processed after the first* – in general, the most recently-processed code block and associated poly-switch takes priority.

We can explore this further by considering the YAML settings in Listing 265; upon running the command

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

we obtain the output given in Listing 264.

LISTING 264: mycommand1.tex
using Listing 265

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}
{
  \mand_arg_text
  \mand_arg_text}
```

LISTING 265: mycom-mlb5.yaml

-m

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

As previously, the most-recently-processed code block takes priority – as before, the second (i.e, *last*) argument. Exploring this further, we consider the YAML settings in Listing 267, which give associated output in Listing 266.

LISTING 266: mycommand1.tex
using Listing 267

```
\mycommand
{
  \mand_arg_text
  \mand_arg_text}%
{
  \mand_arg_text
  \mand_arg_text}
```

LISTING 267: 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 268, noting that it contains nested environments.

LISTING 268: `nested-env.tex`

```
\begin{one}
one_text
\begin{two}
two_text
\end{two}
\end{one}
```

Let's use the YAML settings given in Listing 270, which upon running the command

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

gives the output in Listing 269.

LISTING 269: `nested-env.tex` using Listing 269

```
\begin{one}
  one_text
  \begin{two}
    two_text\end{two}\end{one}
```

LISTING 270: `nested-env-mlb1.yaml`

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

In Listing 269, 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 268, 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 269, this means that during Phase 1:

- the two environment is found first, and the line break ahead of the `\end{two}` statement is removed because `EndStartsOnOwnLine` is set to -1. Importantly, because, *at this stage*, `\end{two}` *does* finish with a line break, `EndFinishesWithLineBreak` causes no action.
- next, the one environment is found; the line break ahead of `\end{one}` is removed because `EndStartsOnOwnLine` is set to -1.

The indentation is done in Phase 2; in Phase 3 *there is no option to add a line break after the end statements*. We can justify this by remembering that during Phase 3, the one environment will be



found and processed first, followed by the two environment. If the two environment were to add a line break after the `\end{two}` statement, then `latexindent.pl` would have no way of knowing how much indentation to add to the subsequent text (in this case, `\end{one}`).

We can explore this further using the poly-switches in Listing 272; upon running the command

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

we obtain the output given in Listing 271.

LISTING 271: `nested-env.tex` using Listing 272

```
\begin{one}
  \one_text
  \begin{two}
    \two_text
  \end{two}\end{one}
```

LISTING 272: `nested-env-mlb2.yaml`

```
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 Conclusions and known limitations

There are a number of known limitations of the script, and almost certainly quite a few that are *unknown*!

For example, `latexindent.pl` will not indent the following code correctly, because of the unmatched `[`. I'm hopeful to be able to resolve this issue in a future version.

```
\parbox{
\@ifnextchar[{\@assignmentwithcutoff}{\@assignmentnocutoff}
}
```

The main other limitation is to do with the alignment routine of environments/commands that contain delimiters which are specified in `lookForAlignDelims`.

The routine works well for 'standard' blocks of code that have the same number of `&` per line, but it will not do anything for lines that do not – such examples include `tabular` environments that use `\multicolumn` or perhaps spread cell contents across multiple lines. For each alignment block (`tabular`, `align`, etc) `latexindent.pl` first of all makes a record of the maximum number of `&`; if each row does not have that number of `&` then it will not try to format that row. Details will be given in `indent.log` assuming that trace mode is active.

You can run `latexindent` on `.sty`, `.cls` and any file types that you specify in `fileExtensionPreference` (see Listing 11 on page 14); if you find a case in which the script struggles, please feel free to report it at [6], and in the meantime, consider using a `noIndentBlock` (see page 16).

I hope that this script is useful to some; if you find an example where the script does not behave as you think it should, the best way to contact me is to report an issue on [6]; otherwise, feel free to find me on the <http://tex.stackexchange.com/users/6621/cmhughes>.



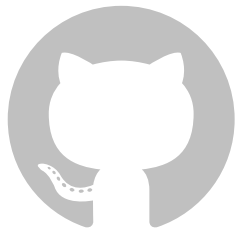
8 References

8.1 External links

- [1] *A Perl script for indenting tex files*. URL: <http://tex.blogoverflow.com/2012/08/a-perl-script-for-indenting-tex-files/> (visited on 01/23/2017).
- [3] *CPAN: Comprehensive Perl Archive Network*. URL: <http://www.cpan.org/> (visited on 01/23/2017).
- [6] *Home of latexindent.pl*. URL: <https://github.com/cmhughes/latexindent.pl> (visited on 01/23/2017).
- [9] *Perlbrew*. URL: <http://perlbrew.pl/> (visited on 01/23/2017).
- [10] *Strawberry Perl*. URL: <http://strawberryperl.com/> (visited on 01/23/2017).
- [11] *Text::Wrap Perl module*. URL: <http://perldoc.perl.org/Text/Wrap.html> (visited on 05/01/2017).
- [12] *Video demonstration of latexindent.pl on youtube*. URL: <https://www.youtube.com/watch?v=wo38aaH2F4E&spfreload=10> (visited on 02/21/2017).

8.2 Contributors

- [2] Paulo Cereda. *arara rule, indent.yaml*. May 23, 2013. URL: <https://github.com/cereda/arara/blob/master/rules/indent.yaml> (visited on 01/23/2017).
- [4] Jacobo Diaz. *Changed shebang to make the script more portable*. July 23, 2014. URL: <https://github.com/cmhughes/latexindent.pl/pull/17> (visited on 01/23/2017).
- [5] Jacobo Diaz. *Hiddenconfig*. July 21, 2014. URL: <https://github.com/cmhughes/latexindent.pl/pull/18> (visited on 01/23/2017).
- [7] Jason Juang. *add in PATH installation*. Nov. 24, 2015. URL: <https://github.com/cmhughes/latexindent.pl/pull/38> (visited on 01/23/2017).
- [8] John Owens. *Paragraph line break routine removal*. May 27, 2017. URL: <https://github.com/cmhughes/latexindent.pl/issues/33> (visited on 05/27/2017).
- [13] 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 273 (`perl helloworld.pl`) then you will be able to run `latexindent.pl`, otherwise you may need to install the missing modules.

LISTING 273: `helloworld.pl`

```
#!/usr/bin/perl

use strict;
use warnings;
use utf8;
use PerlIO::encoding;
use Unicode::GCString;
use open ':std', ':encoding(UTF-8)';
use Text::Wrap;
use FindBin;
use YAML::Tiny;
use File::Copy;
use File::Basename;
use File::HomeDir;
use Getopt::Long;
use Data::Dumper;

print "hello_world";
exit;
```



Installing the modules given in Listing 273 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 [9]; 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 [3].

`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 [6]) that can update the path variables¹¹. If you're on a Linux or Mac machine, then you'll want `CMakeLists.txt` from [6].

B.1 Add to path for Linux

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

1. download `latexindent.pl` and its associated modules, `defaultSettings.yaml`, to your chosen directory from [6] ;
2. within your directory, create a directory called `path-helper-files` and download `CMakeLists.txt` and `cmake_uninstall.cmake.in` from [6]/`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
```

¹¹Thanks to [7] for this feature!



5. run

```
cmh:~$ ls /usr/local/bin
```

again to check that `latexindent.pl`, its modules and `defaultSettings.yaml` have been added.

To remove the files, run

```
cmh:~$ sudo make uninstall}.
```

B.2 Add to path for Windows

To add `latexindent.exe` to the path for Windows, follow these steps:

1. download `latexindent.exe`, `defaultSettings.yaml`, `add-to-path.bat` from [6] to your chosen directory;
2. open a command prompt and run the following command to see what is *currently* in your `%path%` variable;

```
C:\Users\cmh>echo %path%
```

3. right click on `add-to-path.bat` and *Run as administrator*;
4. log out, and log back in;
5. open a command prompt and run

```
C:\Users\cmh>echo %path%
```

to check that the appropriate directory has been added to your `%path%`.

To remove the directory from your `%path%`, run `remove-from-path.bat` as administrator.

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 274 are *obsolete* from Version 3.0 onwards.



LISTING 274: 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 275 and 276

LISTING 275:
indentAfterThisHeading in Version 2.2

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

LISTING 276:
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 277; as of Version 3.0, you would write YAML as in Listing 278 or, if you're using `-m` switch, Listing 279.

LISTING 277: noAdditionalIndent in Version 2.2

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

LISTING 278: noAdditionalIndent for displayMath in Version 3.0

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

LISTING 279: noAdditionalIndent for displayMath in Version 3.0

```
noAdditionalIndent:
  displayMath: 1
```

End