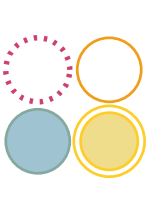
latexindent.pl

Version 3.4.3



Chris Hughes *

June 8, 2018

`latexindent.pl` is a `Perl` script that indents `.tex` (and other) files according to an indentation scheme that the user can modify to suit their taste. Environments, including those with alignment delimiters (such as `tabular`), and commands, including those that can split braces and brackets across lines, are *usually* handled correctly by the script. Options for `verbatim`-like environments and commands, together with indentation after headings (such as `chapter`, `section`, etc) are also available. The script also has the ability to modifiy line breaks, and add comment symbols. All user options are customisable via the switches in the YAML interface; you can find a quick start guide in Section 1.4 on page 7.

Contents

1 Introduction **6**
- 1.1 Thanks . . . 6
- 1.2 License . . . 6
- 1.3 About this documentation . . . 7
- 1.4 Quick start . . . 7

2 Demonstration: before and after **8**

3 How to use the script **9**
- 3.1 From the command line . . . 9
- 3.2 From arara . . . 14

4 indentconfig.yaml, local settings and the -y switch **15**
- 4.1 indentconfig.yaml and .indentconfig.yaml . . . 15
- 4.2 localSettings.yaml . . . 16
- 4.3 The -y|yaml switch . . . 17
- 4.4 Settings load order . . . 17

5 defaultSettings.yaml **18**
- 5.1 The code blocks known latexindent.pl . . . 31
- 5.2 noAdditionalIndent and indentRules . . . 31
  - 5.2.1 Environments and their arguments . . . 33
  - 5.2.2 Environments with items . . . 39
  - 5.2.3 Commands with arguments . . . 40
  - 5.2.4 ifelsefi code blocks . . . 42
  - 5.2.5 specialBeginEnd code blocks . . . 44
  - 5.2.6 afterHeading code blocks . . . 45
  - 5.2.7 The remaining code blocks . . . 47
  - 5.2.8 Summary . . . 49
- 5.3 Commands and the strings between their arguments . . . 49

*and contributors! See Section 8.2 on page 83. For all communication, please visit [6].

6 The -m (modifylinebreaks) switch 54
6.1 oneSentencePerLine: modifying line breaks for sentences . . . 57
6.1.1 sentencesFollow . . . 59
6.1.2 sentencesBeginWith . . . 60
6.1.3 sentencesEndWith . . . 61
6.1.4 Features of the oneSentencePerLine routine . . . 63
6.2 removeParagraphLineBreaks: modifying line breaks for paragraphs . . . 64
6.3 Poly-switches . . . 70
6.4 modifyLineBreaks for environments . . . 71
6.4.1 Adding line breaks: BeginStartsOnOwnLine and BodyStartsOnOwnLine . . . 71
6.4.2 Adding line breaks using EndStartsOnOwnLine and EndFinishesWithLineBreak 72
6.4.3 poly-switches only add line breaks when necessary . . . 73
6.4.4 Removing line breaks (poly-switches set to −1) . . . 74
6.4.5 About trailing horizontal space . . . 75
6.4.6 poly-switch line break removal and blank lines . . . 75
6.5 Poly-switches for other code blocks . . . 77
6.6 Partnering BodyStartsOnOwnLine with argument-based poly-switches . . . 78
6.7 Conflicting poly-switches: sequential code blocks . . . 79
6.8 Conflicting poly-switches: nested code blocks . . . 80

7 Conclusions and known limitations 82

8 References 82
8.1 External links . . . 82
8.2 Contributors . . . 83

A Required Perl modules 83
A.1 Module installer script . . . 84
A.2 Manually installed modules . . . 84

B Updating the path variable 84
B.1 Add to path for Linux . . . 84
B.2 Add to path for Windows . . . 85

C logFilePreferences 86

D Differences from Version 2.2 to 3.0 86

Listings

LISTING 1: `demo-tex.tex` · · · · · 7
LISTING 2: `fileExtensionPreference` · · · · · 7
LISTING 3: `modifyLineBreaks` · · · · · 7
LISTING 4: Possible error messages · · · · · 8
LISTING 5: `filecontents1.tex` · · · · · 8
LISTING 6: `filecontents1.tex` default output · · · · · 8
LISTING 7: `tikzset.tex` · · · · · 8
LISTING 8: `tikzset.tex` default output · · · · · 8
LISTING 9: `pstricks.tex` · · · · · 9
LISTING 10: `pstricks.tex` default output · · · · · 9
LISTING 11: `arara` sample usage · · · · · 14
LISTING 15: `fileExtensionPreference` · · · · · 18
LISTING 16: `logFilePreferences` · · · · · 19
LISTING 17: `verbatimEnvironments` · · · · · 19
LISTING 18: `verbatimCommands` · · · · · 19
LISTING 19: `noIndentBlock` · · · · · 20
LISTING 20: `noIndentBlock` demonstration · · · · · 20
LISTING 21: removeTrailingWhitespace · · · · · 20
LISTING 23: `fileContentsEnvironments` · · · · · 20
LISTING 24: lookForPreamble · · · · · 21
LISTING 25: Motivating `preambleCommandsBeforeEnvironments` 21
LISTING 27: `tabular1.tex` · · · · · 22
LISTING 28: `tabular1.tex` default output · · · · · 22
LISTING 29: `tabular.yaml` · · · · · 22
LISTING 30: `tabular2.tex` · · · · · 23
LISTING 31: `tabular2.yaml` · · · · · 23
LISTING 32: `tabular3.yaml` · · · · · 23
LISTING 33: `tabular4.yaml` · · · · · 23
LISTING 34: `tabular5.yaml` · · · · · 23
LISTING 35: `tabular6.yaml` · · · · · 23
LISTING 36: `tabular7.yaml` · · · · · 23

Listing 37: tabular8.yaml 23
Listing 38: tabular2.tex default output 23
Listing 39: tabular2.tex using Listing 31 24
Listing 40: tabular2.tex using Listing 32 24
Listing 41: tabular2.tex using Listings 31 and 33 ... 24
Listing 42: tabular2.tex using Listings 31 and 34 ... 24
Listing 43: tabular2.tex using Listings 31 and 35 ... 24
Listing 44: tabular2.tex using Listings 31 and 36 ... 25
Listing 45: tabular2.tex using Listings 31 and 37 ... 25
Listing 46: matrix1.tex 26
Listing 47: matrix1.tex default output 26
Listing 48: align-block.tex 26
Listing 49: align-block.tex default output 26
Listing 50: indentAfterItems 26
Listing 51: items1.tex 26
Listing 52: items1.tex default output 26
Listing 53: itemNames 26
Listing 54: specialBeginEnd 27
Listing 55: special1.tex before 27
Listing 56: special1.tex default output 27
Listing 57: specialLR.tex 27
Listing 58: specialsLeftRight.yaml 28
Listing 59: specialBeforeCommand.yaml 28
Listing 60: specialLR.tex using Listing 58 28
Listing 61: specialLR.tex using Listings 58 and 59 . 28
Listing 62: special2.tex 28
Listing 63: middle.yaml 29
Listing 64: special2.tex using Listing 63 29
Listing 65: middle1.yaml 29
Listing 66: special2.tex using Listing 65 29
Listing 67: indentAfterHeadings 29
Listing 68: headings1.yaml 30
Listing 69: headings1.tex 30
Listing 70: headings1.tex using Listing 68 30
Listing 71: headings1.tex second modification 30
Listing 72: mult-nested.tex 31
Listing 73: mult-nested.tex default output 31
Listing 74: max-indentation1.yaml 31
Listing 75: mult-nested.tex using Listing 74 31
Listing 76: myenv.tex 33
Listing 77: myenv-noAdd1.yaml 33
Listing 78: myenv-noAdd2.yaml 33
Listing 79: myenv.tex output (using either Listing 77 or Listing 78) 34
Listing 80: myenv-noAdd3.yaml 34
Listing 81: myenv-noAdd4.yaml 34
Listing 82: myenv.tex output (using either Listing 80 or Listing 81) 34
Listing 83: myenv-args.tex 34
Listing 84: myenv-args.tex using Listing 77 35
Listing 85: myenv-noAdd5.yaml 35
Listing 86: myenv-noAdd6.yaml 35
Listing 87: myenv-args.tex using Listing 85 35
Listing 88: myenv-args.tex using Listing 86 35
Listing 89: myenv-rules1.yaml 36
Listing 90: myenv-rules2.yaml 36
Listing 91: myenv.tex output (using either Listing 89 or Listing 90) 36
Listing 92: myenv-args.tex using Listing 89 36
Listing 93: myenv-rules3.yaml 37
Listing 94: myenv-rules4.yaml 37
Listing 95: myenv-args.tex using Listing 93 37
Listing 96: myenv-args.tex using Listing 94 37
Listing 97: noAdditionalIndentGlobal 37
Listing 98: myenv-args.tex using Listing 97 38
Listing 99: myenv-args.tex using Listings 89 and 97 38
Listing 100: opt-args-no-add-glob.yaml 38
Listing 101: mand-args-no-add-glob.yaml 38
Listing 102: myenv-args.tex using Listing 100 38
Listing 103: myenv-args.tex using Listing 101 38
Listing 104: indentRulesGlobal 38
Listing 105: myenv-args.tex using Listing 104 39
Listing 106: myenv-args.tex using Listings 89 and 104 39
Listing 107: opt-args-indent-rules-glob.yaml .. 39
Listing 108: mand-args-indent-rules-glob.yaml 39
Listing 109: myenv-args.tex using Listing 107 39
Listing 110: myenv-args.tex using Listing 108 39
Listing 111: item-noAdd1.yaml 40
Listing 112: item-rules1.yaml 40
Listing 113: items1.tex using Listing 111 40
Listing 114: items1.tex using Listing 112 40
Listing 115: items-noAdditionalGlobal.yaml 40
Listing 116: items-indentRulesGlobal.yaml 40
Listing 117: mycommand.tex 41
Listing 118: mycommand.tex default output 41
Listing 119: mycommand-noAdd1.yaml 41
Listing 120: mycommand-noAdd2.yaml 41
Listing 121: mycommand.tex using Listing 119 41
Listing 122: mycommand.tex using Listing 120 41
Listing 123: mycommand-noAdd3.yaml 41
Listing 124: mycommand-noAdd4.yaml 41
Listing 125: mycommand.tex using Listing 123 42
Listing 126: mycommand.tex using Listing 124 42
Listing 127: mycommand-noAdd5.yaml 42
Listing 128: mycommand-noAdd6.yaml 42
Listing 129: mycommand.tex using Listing 127 42

LISTING 130: mycommand.tex using Listing 128 42
LISTING 131: ifelsefi1.tex 43
LISTING 132: ifelsefi1.tex default output 43
LISTING 133: ifnum-noAdd.yaml 43
LISTING 134: ifnum-indent-rules.yaml 43
LISTING 135: ifelsefi1.tex using Listing 133 43
LISTING 136: ifelsefi1.tex using Listing 134 43
LISTING 137: ifelsefi-noAdd-glob.yaml 43
LISTING 138: ifelsefi-indent-rules-global.yaml 43
LISTING 139: ifelsefi1.tex using Listing 137 44
LISTING 140: ifelsefi1.tex using Listing 138 44
LISTING 141: ifelsefi2.tex 44
LISTING 142: ifelsefi2.tex default output 44
LISTING 143: displayMath-noAdd.yaml 44
LISTING 144: displayMath-indent-rules.yaml 44
LISTING 145: special1.tex using Listing 143 45
LISTING 146: special1.tex using Listing 144 45
LISTING 147: special-noAdd-glob.yaml 45
LISTING 148: special-indent-rules-global.yaml 45
LISTING 149: special1.tex using Listing 147 45
LISTING 150: special1.tex using Listing 148 45
LISTING 151: headings2.tex 45
LISTING 152: headings2.tex using Listing 153 46
LISTING 153: headings3.yaml 46
LISTING 154: headings2.tex using Listing 155 46
LISTING 155: headings4.yaml 46
LISTING 156: headings2.tex using Listing 157 46
LISTING 157: headings5.yaml 46
LISTING 158: headings2.tex using Listing 159 46
LISTING 159: headings6.yaml 46
LISTING 160: headings2.tex using Listing 161 47
LISTING 161: headings7.yaml 47
LISTING 162: headings2.tex using Listing 163 47
LISTING 163: headings8.yaml 47
LISTING 164: headings2.tex using Listing 165 47
LISTING 165: headings9.yaml 47
LISTING 166: pgfkeys1.tex 47
LISTING 167: pgfkeys1.tex default output 47
LISTING 168: child1.tex 48
LISTING 169: child1.tex default output 48
LISTING 170: psforeach1.tex 48
LISTING 171: psforeach1.tex default output 48
LISTING 172: noAdditionalIndentGlobal 49
LISTING 173: indentRulesGlobal 49
LISTING 174: commandCodeBlocks 49
LISTING 175: pstricks1.tex 50
LISTING 176: pstricks1 default output 50
LISTING 177: pstricks1.tex using Listing 178 50
LISTING 178: noRoundParentheses.yaml 50
LISTING 179: pstricks1.tex using Listing 180 50
LISTING 180: defFunction.yaml 50
LISTING 181: tikz-node1.tex 51
LISTING 182: tikz-node1 default output 51
LISTING 183: tikz-node1.tex using Listing 184 51
LISTING 184: draw.yaml 51
LISTING 185: tikz-node1.tex using Listing 186 52
LISTING 186: no-strings.yaml 52
LISTING 187: amalgamate-demo.yaml 52
LISTING 188: amalgamate-demo1.yaml 52
LISTING 189: amalgamate-demo2.yaml 52
LISTING 190: amalgamate-demo3.yaml 52
LISTING 191: for-each.tex 53
LISTING 192: for-each default output 53
LISTING 193: for-each.tex using Listing 194 53
LISTING 194: foreach.yaml 53
LISTING 195: ifnextchar.tex 53
LISTING 196: ifnextchar.tex default output 53
LISTING 197: ifnextchar.tex using Listing 198 53
LISTING 198: no-ifnextchar.yaml 53
LISTING 199: modifyLineBreaks 54
LISTING 200: mlb1.tex 55
LISTING 201: mlb1.tex out output 55
LISTING 202: textWrapOptions 55
LISTING 203: textwrap1.tex 55
LISTING 204: textwrap1-mod1.tex 56
LISTING 205: textwrap1.yaml 56
LISTING 206: textwrap2.tex 56
LISTING 207: textwrap2-mod1.tex 56
LISTING 208: textwrap3.tex 56
LISTING 209: textwrap3-mod1.tex 57
LISTING 210: textWrapOptions 57
LISTING 211: textwrap4.tex 57
LISTING 212: textwrap4-mod2.tex 57
LISTING 213: textwrap2.yaml 57
LISTING 214: oneSentencePerLine 58
LISTING 215: multiple-sentences.tex 58
LISTING 216: multiple-sentences.tex using Listing 217 59
LISTING 217: manipulate-sentences.yaml 59
LISTING 218: multiple-sentences.tex using Listing 219 59
LISTING 219: keep-sen-line-breaks.yaml 59
LISTING 220: sentencesFollow 59
LISTING 221: sentencesBeginWith 59
LISTING 222: sentencesEndWith 59
LISTING 223: multiple-sentences.tex using Listing 224 60

Listing 224: sentences-follow1.yaml 60
Listing 225: multiple-sentences1.tex 60
Listing 226: multiple-sentences1.tex using Listing 217 on page 59 60
Listing 227: multiple-sentences1.tex using Listing 228 60
Listing 228: sentences-follow2.yaml 60
Listing 229: multiple-sentences2.tex 61
Listing 230: multiple-sentences2.tex using Listing 217 on page 59 61
Listing 231: multiple-sentences2.tex using Listing 232 61
Listing 232: sentences-begin1.yaml 61
Listing 233: multiple-sentences.tex using Listing 234 61
Listing 234: sentences-end1.yaml 61
Listing 235: multiple-sentences.tex using Listing 236 62
Listing 236: sentences-end2.yaml 62
Listing 237: url.tex 62
Listing 238: url.tex using Listing 217 on page 59 62
Listing 239: url.tex using Listing 240 63
Listing 240: alt-full-stop1.yaml 63
Listing 241: multiple-sentences3.tex 63
Listing 242: multiple-sentences3.tex using Listing 217 on page 59 63
Listing 243: multiple-sentences4.tex 64
Listing 244: multiple-sentences4.tex using Listing 217 on page 59 64
Listing 245: multiple-sentences4.tex using Listing 219 on page 59 64
Listing 246: multiple-sentences4.tex using Listing 247 64
Listing 247: item-rules2.yaml 64
Listing 248: removeParagraphLineBreaks 65
Listing 249: shortlines.tex 65
Listing 250: remove-para1.yaml 65
Listing 251: shortlines1.tex 65
Listing 252: shortlines1-tws.tex 66
Listing 253: shortlines-mand.tex 66
Listing 254: shortlines-opt.tex 66
Listing 255: shortlines-mand1.tex 66
Listing 256: shortlines-opt1.tex 66
Listing 257: shortlines-envs.tex 67
Listing 258: remove-para2.yaml 67
Listing 259: remove-para3.yaml 67
Listing 260: shortlines-envs2.tex 67
Listing 261: shortlines-envs3.tex 68
Listing 262: shortlines-md.tex 68
Listing 263: remove-para4.yaml 68
Listing 264: shortlines-md4.tex 69
Listing 265: paragraphsStopAt 69
Listing 266: sl-stop.tex 69
Listing 267: stop-command.yaml 69
Listing 268: stop-comment.yaml 69
Listing 269: sl-stop4.tex 70
Listing 270: sl-stop4-command.tex 70
Listing 271: sl-stop4-comment.tex 70
Listing 272: environments 71
Listing 273: env-mlb1.tex 71
Listing 274: env-mlb1.yaml 71
Listing 275: env-mlb2.yaml 71
Listing 276: env-mlb.tex using Listing 274 71
Listing 277: env-mlb.tex using Listing 275 71
Listing 278: env-mlb3.yaml 72
Listing 279: env-mlb4.yaml 72
Listing 280: env-mlb.tex using Listing 278 72
Listing 281: env-mlb.tex using Listing 279 72
Listing 282: env-mlb5.yaml 72
Listing 283: env-mlb6.yaml 72
Listing 284: env-mlb.tex using Listing 282 72
Listing 285: env-mlb.tex using Listing 283 72
Listing 286: env-mlb7.yaml 72
Listing 287: env-mlb8.yaml 72
Listing 288: env-mlb.tex using Listing 286 72
Listing 289: env-mlb.tex using Listing 287 72
Listing 290: env-mlb9.yaml 73
Listing 291: env-mlb10.yaml 73
Listing 292: env-mlb.tex using Listing 290 73
Listing 293: env-mlb.tex using Listing 291 73
Listing 294: env-mlb11.yaml 73
Listing 295: env-mlb12.yaml 73
Listing 296: env-mlb.tex using Listing 294 73
Listing 297: env-mlb.tex using Listing 295 73
Listing 298: env-mlb2.tex 73
Listing 299: env-mlb3.tex 73
Listing 300: env-mlb3.tex using Listing 275 on page 71 74
Listing 301: env-mlb3.tex using Listing 279 on page 72 74
Listing 302: env-mlb4.tex 74
Listing 303: env-mlb13.yaml 74
Listing 304: env-mlb14.yaml 74
Listing 305: env-mlb15.yaml 74
Listing 306: env-mlb16.yaml 74
Listing 307: env-mlb4.tex using Listing 303 74
Listing 308: env-mlb4.tex using Listing 304 74
Listing 309: env-mlb4.tex using Listing 305 75
Listing 310: env-mlb4.tex using Listing 306 75
Listing 311: env-mlb5.tex 75
Listing 312: removeTWS-before.yaml 75
Listing 313: env-mlb5.tex using Listings 307 to 310 75

LISTING 314: env-mlb5.tex using Listings 307 to 310 and Listing 312 75
LISTING 315: env-mlb6.tex 76
LISTING 316: UnpreserveBlankLines.yaml 76
LISTING 317: env-mlb6.tex using Listings 307 to 310 76
LISTING 318: env-mlb6.tex using Listings 307 to 310 and Listing 316 76
LISTING 319: env-mlb7.tex 76
LISTING 320: env-mlb7-preserve.tex 76
LISTING 321: env-mlb7-no-preserve.tex 77
LISTING 322: mycommand1.tex 78
LISTING 323: mycommand1.tex using Listing 324 78
LISTING 324: mycom-mlb1.yaml 78
LISTING 325: mycommand1.tex using Listing 326 79
LISTING 326: mycom-mlb2.yaml 79
LISTING 327: mycommand1.tex using Listing 328 79
LISTING 328: mycom-mlb3.yaml 79
LISTING 329: mycommand1.tex using Listing 330 79
LISTING 330: mycom-mlb4.yaml 79
LISTING 331: mycommand1.tex using Listing 332 80
LISTING 332: mycom-mlb5.yaml 80
LISTING 333: mycommand1.tex using Listing 334 80
LISTING 334: mycom-mlb6.yaml 80
LISTING 335: nested-env.tex 80
LISTING 336: nested-env.tex using Listing 337 80
LISTING 337: nested-env-mlb1.yaml 80
LISTING 338: nested-env.tex using Listing 339 81
LISTING 339: nested-env-mlb2.yaml 81
LISTING 340: matrix2.tex 82
LISTING 341: matrix2.tex default output 82
LISTING 342: helloworld.pl 83
LISTING 343: simple.tex 86
LISTING 344: logfile-prefs1.yaml 86
LISTING 345: indent.log 86
LISTING 346: Obsolete YAML fields from Version 3.0 87
LISTING 347: indentAfterThisHeading in Version 2.2 87
LISTING 348: indentAfterThisHeading in Version 3.0 87
LISTING 349: noAdditionalIndent in Version 2.2 87
LISTING 350: noAdditionalIndent for displayMath in Version 3.0 87
LISTING 351: noAdditionalIndent for displayMath in Version 3.0 87

1 Introduction

1.1 Thanks

I first created `latexindent.pl` to help me format chapter files in a big project. After I blogged about it on the TEX stack exchange [1] I received some positive feedback and follow-up feature requests. A big thank you to Harish Kumar [8] 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 83 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 Section 5) 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 D on page 86 and the comments throughout this document for details.*

1.3 About this documentation

As you read through this documentation, you will see many listings; in this version of the documentation, there are a total of 351. This may seem a lot, but I deem it necessary in presenting the various different options of `latexindent.pl` and the associated output that they are capable of producing.

The different listings are presented using different styles:

LISTING 1: demo-tex.tex
```
demonstration .tex file
```

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

LISTING 2: fileExtensionPreference
```
38 fileExtensionPreference:
39     .tex: 1
40     .sty: 2
41     .cls: 3
42     .bib: 4
```

This type of listing is a `.yaml` file; when you see line numbers given (as here) it means that the snippet is taken directly from `defaultSettings.yaml`, discussed in detail in Section 5 on page 18.

LISTING 3: modifyLineBreaks -m
```
394 modifyLineBreaks:
395     preserveBlankLines: 1
396     condenseMultipleBlankLinesInto: 1
```

This type of listing is a `.yaml` file, but it will only be relevant when the `-m` switch is active; see Section 6 on page 54 for more details.

N: 2017-06-25

You will occasionally see dates shown in the margin (for example, next to this paragraph!) which detail the date of the version in which the feature was implemented; the 'N' stands for 'new as of the date shown' and 'U' stands for 'updated as of the date shown'. If you see ✨, it means that the feature is either new (N) or updated (U) as of the release of the current version; if you see ✨ attached to a listing, then it means that listing is new (N) or updated (U) as of the current version. If you have not read this document before (and even if you have!), then you can ignore every occurrence of the ✨; they are simply there to highlight new and updated features. The new and updated features in this documentation (V3.4.3) are on the following pages:

update to specialBeginEnd (N) 28
updates to ifElseFi code blocks (U) 44
commandCodeBlocks (U) 49
amalgamate feature in commandCodeBlocks (U) 52
commandNameSpecial (U) 53
new ifElseFi code block polyswitches (N) 77
new special code block polyswitches (N) 78

1.4 Quick start

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

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

from the command line. If you receive an error message such as that given in Listing 4, then you need to install the missing perl modules.

LISTING 4: Possible error messages

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

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

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

You might also like to see https://stackoverflow.com/questions/19590042/error-cant-locate-file-homedir-pm-in-inc, for example, as well as appendix A on page 83.

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 [16]

As you look at Listings 5 to 10, 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 5 to 10 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 5: filecontents1.tex

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

LISTING 6: filecontents1.tex default output

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

LISTING 7: tikzset.tex

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

LISTING 8: tikzset.tex default output

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

LISTING 9: 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 10: pstricks.tex default output

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

3 How to use the script

latexindent.pl ships as part of the TEXLive distribution for Linux and Mac users; latexindent.exe ships as part of the TEXLive and MiKTEX distributions for Windows users. These files are also available from github [6] should you wish to use them without a TEX distribution; in this case, you may like to read appendix B on page 84 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 18.

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 83 for details; in particular, note that a module installer helper script is shipped with latexindent.pl.

N: 2018-01-13

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.

N: 2017-06-25

-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 exists[1]. Note that if `latexindent.pl` is called with both the `-w` and `-o` switches, then `-w` will be ignored and `-o` will take priority (this seems safer than the other way round).

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

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

N: 2017-06-25

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

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

N: 2017-06-25

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

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

N: 2017-06-25

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

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

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

Calling `latexindent.pl` with simply

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

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

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

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

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

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

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

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

See appendix D on page 86 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 both *relative* and *absolute paths* for your YAML files; for example

U: 2017-08-21

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

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

N: 2017-06-25

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

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

which translate, respectively, to

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

Note that the following is *not* allowed:

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

and

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

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

N: 2017-06-25

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

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

`-y, -yaml=yaml settings`

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

N: 2017-08-21

You can specify YAML settings from the command line using the `-y` or `-yaml` switch; sample demonstrations are given above. Note, in particular, that multiple settings can be specified by separating them via commas. There is a further option to use a `;` to separate fields, which is demonstrated in Section 4.3 on page 17.

Any settings specified via this switch will be loaded *after* any specified using the `-l` switch. This is discussed further in Section 4.4 on page 17.

`-d, -onlydefault`

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

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

U: 2017-08-21

`-c, -cruft=<directory>`

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

If you wish to have backup files and `indent.log` written to a directory other than the current working directory, then you can send these 'cruft' files to another directory.

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

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

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

`-sl, -screenlog`

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

N: 2018-01-13

Using this option tells `latexindent.pl` to output the log file to the screen, as well as to your chosen log file.

`-m, -modifylinebreaks`

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

One of the most exciting developments in Version 3.0 is the ability to modify line breaks; for full details see Section 6 on page 54

`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 15 on page 18 for full details.

`STDIN`

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

N: 2018-01-13

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

U: 2018-01-13

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

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

Once you have finished typing your input, you can press

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

to signify that your input has finished.

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`; you can find the arara rule at [2].

You can use the rule in any of the ways described in Listing 11 (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 11: `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.

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

TABLE 1: `arara` directive arguments and corresponding switches

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

4 indentconfig.yaml, local settings and the -y switch

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

4.1 indentconfig.yaml and .indentconfig.yaml

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

For Mac and Linux users, their home directory is `/username` while Windows (Vista onwards) is `C:\Users\username`[2] Listing 12 shows a sample `indentconfig.yaml` file.

LISTING 12: `indentconfig.yaml` (sample)

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

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

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

To get started with your own settings file, you might like to save a copy of `defaultSettings.yaml` in another directory and call it, for example, `mysettings.yaml`. Once you have added the path to `indentconfig.yaml` you can change the switches and add more code-block names to it as you see fit – have a look at Listing 13 for an example that uses four tabs for the default indent, adds the `tabbing` environment/command to the list of environments that contains alignment delimiters; you might also like to refer to the many YAML files detailed throughout the rest of this documentation.

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

LISTING 13: `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`.

4.2 localSettings.yaml

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

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

then `latexindent.pl` will (assuming it exists) load `localSettings.yaml` from the same directory as `myfile.tex`.

If you'd prefer to name your `localSettings.yaml` file something different, (say, `mysettings.yaml` as in Listing 13) then you can call `latexindent.pl` using, for example,

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

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

Your settings file can contain any switches that you'd like to change; a sample is shown in Listing 14, and you'll find plenty of further examples throughout this manual.

LISTING 14: `localSettings.yaml` (example)

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

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

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

4.3 The -y|yaml switch

N: 2017-08-21

You may use the `-y` switch to load your settings; for example, if you wished to specify the settings from Listing 14 using the `-y` switch, then you could use the following command:

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

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

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

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

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

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

4.4 Settings load order

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

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

U: 2017-08-21

4. any settings specified in the `-y` switch.

N: 2017-08-21

A visual representation of this is given in Figure 1.

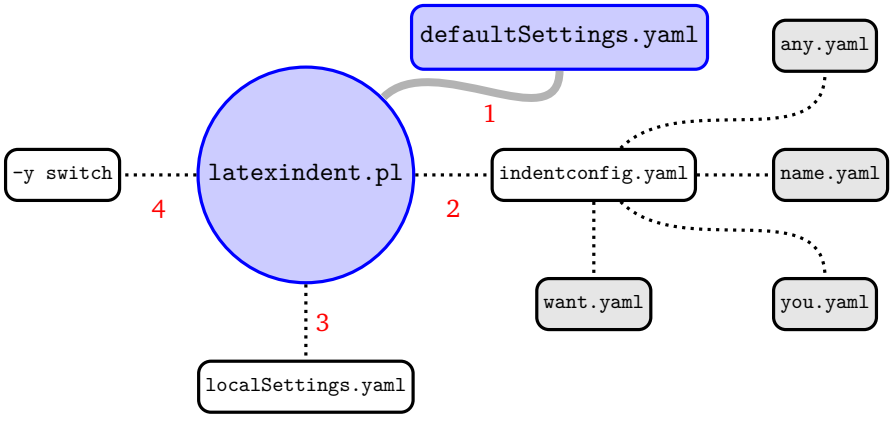


FIGURE 1: Schematic of the load order described in Section 4.4; solid lines represent mandatory files, dotted lines represent optional files. `indentconfig.yaml` can contain as many files as you like. The files will be loaded in order; if you specify settings for the same field in more than one file,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

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 15 means that the script will first look for `myfile.tex`, then `myfile.sty`, `myfile.cls`, and finally `myfile.bib` in order[4].

LISTING 15: `fileExtensionPreference`

```
38 fileExtensionPreference:
39     .tex: 1
40     .sty: 2
41     .cls: 3
42     .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

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

only the most recent; for example, with `maxNumberOfBackUps: 4`, and `cycleThroughBackUps` set to 1 then the `copy` procedure given below would be obeyed.

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

The default value of `cycleThroughBackUps` is 0.

`logFilePreferences`: ⟨*fields*⟩

`latexindent.pl` writes information to `indent.log`, some of which can be customized by changing `logFilePreferences`; see Listing 16. If you load your own user settings (see Section 4 on page 15) then `latexindent.pl` will detail them in `indent.log`; you can choose not to have the details logged by switching `showEveryYamlRead` to 0. Once all of your settings have been loaded, you can see the amalgamated settings in the log file by switching `showAmalgamatedSettings` to 1, if you wish.

LISTING 16: `logFilePreferences`

```
79 logFilePreferences:
80     showEveryYamlRead: 1
81     showAmalgamatedSettings: 0
82     showDecorationStartCodeBlockTrace: 0
83     showDecorationFinishCodeBlockTrace: 0
84     endLogFileWith: '--------------'
85     showGitHubInfoFooter: 1
86     PatternLayout:
87         default: "%A%n"
88         trace: "%A%n"
89         ttrace: "%A%n"
```

N: 2018-01-13

When either of the `trace` modes (see Section 3.1) are active, you will receive detailed information in `indent.log`. You can specify character strings to appear before and after the notification of a found code block using, respectively, `showDecorationStartCodeBlockTrace` and `showDecorationFinishCodeBlockTra` A demonstration is given in appendix C on page 86.

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.

N: 2018-01-13

`latexindent.pl` uses the `log4perl` module [9] to handle the creation of the logfile. You can specify the layout of the information given in the logfile using any of the `Log Layouts` detailed at [9].

`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 17.

LISTING 17: `verbatimEnvironments`

```
93 verbatimEnvironments:
94     verbatim: 1
95     lstlisting: 1
96     minted: 1
```

LISTING 18: `verbatimCommands`

```
99  verbatimCommands:
100     verb: 1
101     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 54).

`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 19.

LISTING 19: noIndentBlock

```
107 noIndentBlock:
108     noindent: 1
109     cmhtest: 1
```

Of course, you don't want to have to specify these as null environments in your code, so you use them with a comment symbol, %, followed by as many spaces (possibly none) as you like; see Listing 20 for example.

LISTING 20: 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 21; each of the fields can take the values 0 or 1. See Listings 312 to 314 on page 75 for before and after results. Thanks to [17] for providing this feature.

LISTING 21: removeTrailingWhitespace

```
112 removeTrailingWhitespace:
113     beforeProcessing: 0
114     afterProcessing: 1
```

You can specify `removeTrailingWhitespace` simply as 0 or 1, if you wish; in this case, `latexindent.pl` will set both `beforeProcessing` and `afterProcessing` to the value you specify; see Listing 22.

N: 2017-06-28

LISTING 22: removeTrailingWhitespace (alt)

```
removeTrailingWhitespace: 1
```

`fileContentsEnvironments`: ⟨*field*⟩

Before `latexindent.pl` determines the difference between preamble (if any) and the main document, it first searches for any of the environments specified in `fileContentsEnvironments`, see Listing 23. The behaviour of `latexindent.pl` on these environments is determined by their location (preamble or not), and the value `indentPreamble`, discussed next.

LISTING 23: fileContentsEnvironments

```
118 fileContentsEnvironments:
119     filecontents: 1
120     filecontents*: 1
```

`indentPreamble`: **0|1**

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

`lookForPreamble`: ⟨*fields*⟩

Not all files contain preamble; for example, `sty`, `cls` and `bib` files typically do *not*. Referencing Listing 24, if you set, for example, `.tex` to `0`, then regardless of the setting of the value of `indentPreamble`, preamble will not be assumed when operating upon `.tex` files.

LISTING 24: lookForPreamble

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

`preambleCommandsBeforeEnvironments`: **0|1**

Assuming that `latexindent.pl` is asked to operate upon the preamble of a document, when this switch is set to `0` then environment code blocks will be sought first, and then command code blocks. When this switch is set to `1`, commands will be sought first. The example that first motivated this switch contained the code given in Listing 25.

LISTING 25: 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 31 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 26). In fact, the fields in `lookForAlignDelims` can actually take two different forms: the *basic* version is shown in Listing 26 and the *advanced* version in Listing 29; we will discuss each in turn.

LISTING 26: lookForAlignDelims (basic)

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

The environments specified in this field will be operated on in a special way by `latexindent.pl`. In particular, it will try and align each column by its alignment tabs. It does have some limitations (discussed further in Section 7), but in many cases it will produce results such as those in Listings 27 and 28.

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 19 on page 20).

LISTING 27: `tabular1.tex`

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

LISTING 28: `tabular1.tex` default output

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

If, for example, you wish to remove the alignment of the `\\` within a delimiter-aligned block, then the advanced form of `lookForAlignDelims` shown in Listing 29 is for you.

LISTING 29: `tabular.yaml`

```
lookForAlignDelims:
   tabular:
      delims: 1
      alignDoubleBackSlash: 0
      spacesBeforeDoubleBackSlash: 0
      multiColumnGrouping: 0
      alignRowsWithoutMaxDelims: 1
      spacesBeforeAmpersand: 1
      spacesAfterAmpersand: 1
   tabularx:
      delims: 1
   longtable: 1
```

Note that you can use a mixture of the basic and advanced form: in Listing 29 `tabular` and `tabularx` are advanced and `longtable` is basic. When using the advanced form, each field should receive at least 1 sub-field, and *can* (but does not have to) receive any of the following fields:

- `delims`: binary switch (0 or 1) equivalent to simply specifying, for example, `tabular: 1` in the basic version shown in Listing 26. If `delims` is set to 0 then the align at ampersand routine will not be called for this code block (default: 1);
- `alignDoubleBackSlash`: binary switch (0 or 1) to determine if `\\` should be aligned (default: 1);

U: 2018-01-13

- `spacesBeforeDoubleBackSlash`: optionally, specifies the number (integer ≥ 0) of spaces to be inserted before `\\` (default: 1). [5]

N: 2017-06-19

- `multiColumnGrouping`: binary switch (0 or 1) that details if `latexindent.pl` should group columns above and below a `\multicolumn` command (default: 0);

N: 2017-06-19

- `alignRowsWithoutMaxDelims`: binary switch (0 or 1) that details if rows that do not contain the maximum number of delimeters should be formatted so as to have the ampersands aligned (default: 1);

N: 2018-01-13

- `spacesBeforeAmpersand`: optionally specifies the number (integer ≥ 0) of spaces to be placed *before* ampersands (default: 1);

N: 2018-01-13

- `spacesAfterAmpersand`: optionally specifies the number (integer ≥ 0) of spaces to be placed *After* ampersands (default: 1);

N: 2018-01-13

- `justification`: optionally specifies the justification of each cell as either *left* or *right* (default: left).

We will explore each of these features using the file `tabular2.tex` in Listing 30 (which contains a `\multicolumn` command), and the YAML files in Listings 31 to 37.

[5] Previously this only activated if `alignDoubleBackSlash` was set to 0.

LISTING 30: 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 31: tabular2.yaml

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

LISTING 32: tabular3.yaml

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

LISTING 33: tabular4.yaml

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

LISTING 34: tabular5.yaml

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

LISTING 35: tabular6.yaml

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

LISTING 36: tabular7.yaml

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

LISTING 37: tabular8.yaml

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

On running the commands

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

we obtain the respective outputs given in Listings 38 to 45.

LISTING 38: tabular2.tex default output

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

LISTING 39: tabular2.tex using Listing 31

```
\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 40: tabular2.tex using Listing 32

```
\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 41: tabular2.tex using Listings 31 and 33

```
\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 42: tabular2.tex using Listings 31 and 34

```
\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 43: tabular2.tex using Listings 31 and 35

```
\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 44: tabular2.tex using Listings 31 and 36

```
\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 45: tabular2.tex using Listings 31 and 37

```
\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 38 and 39 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 38 the columns have been aligned at the ampersand;
- in Listing 39 the `\multicolumn` command has grouped the 2 columns beneath *and* above it, because `multiColumnGrouping` is set to 1 in Listing 31;
- in Listing 40 rows 3 and 6 have *not* been aligned at the ampersand, because `alignRowsWithoutMaxDelims` has been to set to 0 in Listing 32; however, the `\\` *have* still been aligned;
- in Listing 41 the columns beneath and above the `\multicolumn` commands have been grouped (because `multiColumnGrouping` is set to 1), and there are at least 4 spaces *before* each aligned ampersand because `spacesBeforeAmpersand` is set to 4;
- in Listing 42 the columns beneath and above the `\multicolumn` commands have been grouped (because `multiColumnGrouping` is set to 1), and there are at least 4 spaces *after* each aligned ampersand because `spacesAfterAmpersand` is set to 4;
- in Listing 43 the `\\` have *not* been aligned, because `alignDoubleBackSlash` is set to `0`, otherwise the output is the same as Listing 39;
- in Listing 44 the `\\` *have* been aligned, and because `spacesBeforeDoubleBackSlash` is set to `0`, there are no spaces ahead of them; the output is otherwise the same as Listing 39.
- in Listing 45 the cells have been *right*-justified; note that cells above and below the `\multicol` statements have still been group correctly, because of the settings in Listing 31.

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 Section 5); 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 46 and 47 are achievable by default.

LISTING 46: matrix1.tex

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

LISTING 47: 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 48; the default output is shown in Listing 49. 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 48: align-block.tex

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

LISTING 49: align-block.tex default output

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

With reference to Table 2 on page 32 and the, yet undiscussed, fields of `noAdditionalIndent` and `indentRules` (see Section 5.2 on page 31), 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 51 and 52

LISTING 50: indentAfterItems

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

LISTING 51: 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 52: items1.tex default output

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

`itemNames`: ⟨*fields*⟩

If you have your own `item` commands (perhaps you prefer to use `myitem`, for example) then you can put populate them in `itemNames`. For example, users of the `exam` document class might like to add `parts` to `indentAfterItems` and `part` to `itemNames` to their user settings (see Section 4 on page 15 for details of how to configure user settings, and Listing 13 on page 16 in particular .)

LISTING 53: itemNames

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

`specialBeginEnd`: ⟨*fields*⟩

U: 2017-08-21

The fields specified in `specialBeginEnd` are, in their default state, focused on math mode begin and end statements, but there is no requirement for this to be the case; Listing 54 shows the default settings of `specialBeginEnd`.

LISTING 54: `specialBeginEnd`

```
specialBeginEnd:
    displayMath:
        begin: '\\\['
        end: '\\\]'
        lookForThis: 1
    inlineMath:
        begin: '(?<!\$)(?<!\\)\$(?!\$)'
        end: '(?<!\\)\$(?!\$)'
        lookForThis: 1
    displayMathTeX:
        begin: '\$\$'
        end: '\$\$'
        lookForThis: 1
    specialBeforeCommand: 0
```

The field `displayMath` represents `\[...\]`, `inlineMath` represents `$...$` and `displayMathTex` represents `$$...$$`. You can, of course, rename these in your own YAML files (see Section 4.2 on page 16); 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 55 and 56.

LISTING 55: `special1.tex` before

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

LISTING 56: `special1.tex` default output

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

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

N: 2017-08-21

There are examples in which it is advantageous to search for `specialBeginEnd` fields *before* searching for commands, and the `specialBeforeCommand` switch controls this behaviour. For example, consider the file shown in Listing 57.

LISTING 57: `specialLR.tex`

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

Now consider the YAML files shown in Listings 58 and 59

LISTING 58: specialsLeftRight.yaml

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

LISTING 59: specialBeforeCommand.yaml

```
specialBeginEnd:
    specialBeforeCommand: 1
```

Upon running the following commands

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

we receive the respective outputs in Listings 60 and 61.

LISTING 60: specialLR.tex using Listing 58

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

LISTING 61: specialLR.tex using Listings 58 and 59

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

Notice that in:

- Listing 60 the `\left` has been treated as a *command*, with one optional argument;
- Listing 61 the `specialBeginEnd` pattern in Listing 58 has been obeyed because Listing 59 specifies that the `specialBeginEnd` should be sought *before* commands.

N: 2018-04-27

You can,optionally, specify the `middle` field for anything that you specify in `specialBeginEnd`. For example, let's consider the `.tex` file in Listing 62.

LISTING 62: special2.tex

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

Upon saving the YAML settings in Listings 63 and 65 and running the commands

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

then we obtain the output given in Listings 64 and 66.

LISTING 63: middle.yaml

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

LISTING 64: special2.tex using Listing 63

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

LISTING 65: middle1.yaml

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

LISTING 66: special2.tex using Listing 65

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

We note that:

- in Listing 64 the bodies of each of the `Elsif` statements have been indented appropriately;
- the `Else` statement has *not* been indented appropriately in Listing 64 – read on!
- we have specified multiple settings for the `middle` field using the syntax demonstrated in Listing 65 so that the body of the `Else` statement has been indented appropriately in Listing 66.

`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.[6]

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.

LISTING 67: indentAfterHeadings

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

[6] There is a slight difference in interface for this field when comparing Version 2.2 to Version 3.0; see appendix D on page 86 for details.

You can add any of your own custom heading commands to this field, specifying the `level` as appropriate. You can also specify your own indentation in `indentRules` (see Section 5.2 on the next page); you will find the default `indentRules` contains `chapter: " "` which tells `latexindent.pl` simply to use a space character after headings (once `indent` is set to 1 for `chapter`).

For example, assuming that you have the code in Listing 68 saved into `headings1.yaml`, and that you have the text from Listing 69 saved into `headings1.tex`.

LISTING 68: `headings1.yaml`

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

LISTING 69: `headings1.tex`

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

If you run the command

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

then you should receive the output given in Listing 70.

LISTING 70: `headings1.tex` using Listing 68

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

LISTING 71: `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 68 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 71; notice that the `paragraph` and `subsection` are at the same indentation level.

`maximumIndentation`: ⟨*horizontal space*⟩

N: 2017-08-21

You can control the maximum indentation given to your file by specifying the `maximumIndentation` field as horizontal space (but *not* including tabs). This feature uses the `Text::Tabs` module [14], and is *off* by default.

For example, consider the example shown in Listing 72 together with the default output shown in Listing 73.

LISTING 72: mult-nested.tex

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

LISTING 73: mult-nested.tex default output

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

Now say that, for example, you have the `max-indentation1.yaml` from Listing 74 and that you run the following command:

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

You should receive the output shown in Listing 75.

LISTING 74: max-indentation1.yaml

```
maximumIndentation: " "
```

LISTING 75: mult-nested.tex using Listing 74

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

Comparing the output in Listings 73 and 75 we notice that the (default) tabs of indentation have been replaced by a single space.

In general, when using the `maximumIndentation` feature, any leading tabs will be replaced by equivalent spaces except, of course, those found in `verbatimEnvironments` (see Listing 17 on page 19) or `noIndentBlock` (see Listing 19 on page 20).

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.

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; for each type of code block in Table 2 on the next page (which we will call a ⟨*thing*⟩ in what follows) it searches YAML fields for information in the following order:

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

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

Code block	characters allowed in name	example

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

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

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

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

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

items	User specified, see Listings 50 and 53 on page 26	`\begin{enumerate}`
		`  \item ...`
		`\end{enumerate}`

specialBeginEnd	User specified, see Listing 54 on page 27	`\[`
		`  ...`
		`\]`

afterHeading	User specified, see Listing 67 on page 29	`\chapter{title}`
		`  ...`
		`\section{title}`

filecontents	User specified, see Listing 23 on page 20	`\begin{filecontents}`
		`...`
		`\end{filecontents}`

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

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

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

5.2.1 Environments and their arguments . . . 33
5.2.2 Environments with items . . . 39
5.2.3 Commands with arguments . . . 40
5.2.4 ifelsefi code blocks . . . 42
5.2.5 specialBeginEnd code blocks . . . 44
5.2.6 afterHeading code blocks . . . 45
5.2.7 The remaining code blocks . . . 47
keyEqualsValuesBracesBrackets . . . 47
namedGroupingBracesBrackets . . . 48
UnNamedGroupingBracesBrackets . . . 48
filecontents . . . 49
5.2.8 Summary . . . 49

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 76.

LISTING 76: `myenv.tex`

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

`noAdditionalIndent`: ⟨*fields*⟩

If we do not wish `myenv` to receive any additional indentation, we have a few choices available to us, as demonstrated in Listings 77 and 78.

LISTING 77: `myenv-noAdd1.yaml`

```
noAdditionalIndent:
    myenv: 1
```

LISTING 78: `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 79; note in particular that the environment myenv has not received any *additional* indentation, but that the outer environment *has* still received indentation.

LISTING 79: myenv.tex output (using either Listing 77 or Listing 78)

```
\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 80 and 81, 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 82.

LISTING 80: myenv-noAdd3.yaml

```
noAdditionalIndent:
    myenv: 0
```

LISTING 81: myenv-noAdd4.yaml

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

LISTING 82: myenv.tex output (using either Listing 80 or Listing 81)

```
\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 83.

LISTING 83: 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 84; 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 77), then *all* parts of the environment (body, optional and mandatory arguments) are assumed to want no additional indent.

LISTING 84: `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}
```

We may customise `noAdditionalIndent` for optional and mandatory arguments of the `myenv` environment, as shown in, for example, Listings 85 and 86.

LISTING 85: `myenv-noAdd5.yaml`

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

LISTING 86: `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 87 and 88. Note that in Listing 87 the text for the *optional* argument has not received any additional indentation, and that in Listing 88 the *mandatory* argument has not received any additional indentation; in both cases, the *body* has not received any additional indentation.

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

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

LISTING 88: `myenv-args.tex` using Listing 86

```
\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 89 and 90.

LISTING 89: `myenv-rules1.yaml`

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

LISTING 90: `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 91; note in particular that the environment `myenv` has received one tab (from the `outer` environment) plus three spaces from Listing 89 or 90.

LISTING 91: `myenv.tex` output (using either Listing 89 or Listing 90)

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

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

Returning to the example in Listing 83 that contains optional and mandatory arguments. Upon using Listing 89 as in

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

we obtain the output in Listing 92; note that the body, optional argument and mandatory argument of `myenv` have *all* received the same customised indentation.

LISTING 92: `myenv-args.tex` using Listing 89

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

You can specify different indentation rules for the different features using, for example, Listings 93 and 94

LISTING 93: myenv-rules3.yaml

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

LISTING 94: 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 95 and 96.

LISTING 95: myenv-args.tex using Listing 93

```
\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 96: myenv-args.tex using Listing 94

```
\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 95, 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 96, 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 97). Let's say that you change the value of environments to 1 in Listing 97, and that you run

LISTING 97: noAdditionalIndentGlobal

```
279 noAdditionalIndentGlobal:
280     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 98 and 99; in Listing 98 notice that *both* environments receive no additional indentation but that the arguments of myenv still *do* receive indentation. In Listing 99 notice that the *outer* environment does not receive additional indentation, but because of the settings from myenv-rules1.yaml (in Listing 89 on the previous page), the myenv environment still *does* receive indentation.

LISTING 98: myenv-args.tex using Listing 97

```
\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 99: myenv-args.tex using Listings 89 and 97

```
\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 100 and 101

LISTING 100: opt-args-no-add-glob.yaml

```
noAdditionalIndentGlobal:
    optionalArguments: 1
```

LISTING 101: 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 102 and 103. Notice that in Listing 102 the *optional* argument has not received any additional indentation, and in Listing 103 the *mandatory* argument has not received any additional indentation.

LISTING 102: myenv-args.tex using Listing 100

```
\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 103: myenv-args.tex using Listing 101

```
\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 104; if you change the `environments` field to anything involving horizontal space, say `" "`, and then run the following commands

LISTING 104: indentRulesGlobal

```
295 indentRulesGlobal:
296     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 105 and 106. Note that in Listing 105, both the environment blocks have received a single-space indentation, whereas in Listing 106 the `outer` environment has received single-space indentation (specified by `indentRulesGlobal`), but `myenv` has received `"  "`, as specified by the particular `indentRules` for `myenv` Listing 89 on page 36.

LISTING 105: `myenv-args.tex` using Listing 104

```
\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 106: `myenv-args.tex` using Listings 89 and 104

```
\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 107 and 108

LISTING 107: `opt-args-indent-rules-glob.yaml`

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

LISTING 108: `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 109 and 110. Note that the *optional* argument in Listing 109 has received two tabs worth of indentation, while the *mandatory* argument has done so in Listing 110.

LISTING 109: `myenv-args.tex` using Listing 107

```
\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 110: `myenv-args.tex` using Listing 108

```
\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 50 and 53 on page 26, some commands may contain `item` commands; for the purposes of this discussion, we will use the code from Listing 51 on page 26.

Assuming that you've populated `itemNames` with the name of your `item`, you can put the item name into `noAdditionalIndent` as in Listing 111, 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 112

LISTING 111: item-noAdd1.yaml

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

LISTING 112: 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 113 and 114; note that in Listing 113 that the text after each `item` has not received any additional indentation, and in Listing 114, the text after each `item` has received a single space of indentation, specified by Listing 112.

LISTING 113: items1.tex using Listing 111

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

LISTING 114: items1.tex using Listing 112

```
\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 115 and 116. Note that there is a need to 'reset/remove' the `item` field from `indentRules` in both cases (see the hierarchy description given on Section 5.2) as the `item` command is a member of `indentRules` by default.

LISTING 115: items-noAdditionalGlobal.yaml

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

LISTING 116: 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 113 and 114 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 117; when `latexindent.pl` operates on this file, the default output is shown in Listing 118. [7]

[7] The command code blocks have quite a few subtleties, described in Section 5.3 on page 49.

LISTING 117: mycommand.tex

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

LISTING 118: 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 77 and 78 on page 33) we may specify `noAdditionalIndent` either in 'scalar' form, or in 'field' form, as shown in Listings 119 and 120

LISTING 119: mycommand-noAdd1.yaml

```
noAdditionalIndent:
    mycommand: 1
```

LISTING 120: 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 121 and 122

LISTING 121: mycommand.tex using Listing 119

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

LISTING 122: mycommand.tex using Listing 120

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

Note that in Listing 121 that the 'body', optional argument *and* mandatory argument have *all* received no additional indentation, while in Listing 122, 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 85 and 86 on page 35; explicit examples are given in Listings 123 and 124.

LISTING 123: mycommand-noAdd3.yaml

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

LISTING 124: 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 125 and 126.

LISTING 125: mycommand.tex using Listing 123

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

LISTING 126: mycommand.tex using Listing 124

```
\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 125 and 126 has received no additional indent, even though `body` is explicitly set to `0` in both Listings 123 and 124. This is because, by default, `noAdditionalIndentGlobal` for `commands` is set to `1` by default; this can be easily fixed as in Listings 127 and 128.

LISTING 127: mycommand-noAdd5.yaml

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

LISTING 128: 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 129 and 130.

LISTING 129: mycommand.tex using Listing 127

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

LISTING 130: mycommand.tex using Listing 128

```
\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 93 and 94 on page 37 and Listings 104, 107 and 108 on pages 38–39.

5.2.4 ifelsefi code blocks

Let's use the simple example shown in Listing 131; when `latexindent.pl` operates on this file, the output as in Listing 132; note that the body of each of the `\if` statements have been indented, and that the `\else` statement has been accounted for correctly.

LISTING 131: ifelsefi1.tex

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

LISTING 132: ifelsefi1.tex default output

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

It is recommended to specify `noAdditionalIndent` and `indentRules` in the 'scalar' form only for these type of code blocks, although the 'field' form would work, assuming that `body` was specified. Examples are shown in Listings 133 and 134.

LISTING 133: ifnum-noAdd.yaml

```
noAdditionalIndent:
    ifnum: 1
```

LISTING 134: ifnum-indent-rules.yaml

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

After running the following commands,

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

we receive the respective output given in Listings 135 and 136; note that in Listing 135, the `ifnum` code block has *not* received any additional indentation, while in Listing 136, the `ifnum` code block has received one tab and two spaces of indentation.

LISTING 135: ifelsefi1.tex using Listing 133

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

LISTING 136: ifelsefi1.tex using Listing 134

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

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 137 and 138.

LISTING 137: ifelsefi-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
    ifElseFi: 1
```

LISTING 138: ifelsefi-indent-rules-global.yaml

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

Upon running the following commands

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

we receive the outputs in Listings 139 and 140; notice that in Listing 139 neither of the `ifelsefi` code blocks have received indentation, while in Listing 140 both code blocks have received a single space of indentation.

LISTING 139: ifelsefi1.tex using Listing 137

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

LISTING 140: ifelsefi1.tex using Listing 138

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

U: 2018-04-27

We can further explore the treatment of `ifElseFi` code blocks in Listing 141, and the associated default output given in Listing 142; note, in particular, that the bodies of each of the 'or statements' have been indented.

LISTING 141: ifelsefi2.tex

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

LISTING 142: ifelsefi2.tex default output

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

5.2.5 specialBeginEnd code blocks

Let's use the example from Listing 55 on page 27 which has default output shown in Listing 56 on page 27.

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 143 and 144.

LISTING 143: displayMath-noAdd.yaml

```
noAdditionalIndent:
    displayMath: 1
```

LISTING 144: 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 145 and 146; note that in Listing 145, the `displayMath` code block has *not* received any additional indentation, while in Listing 146, the `displayMath` code block has received three tabs worth of indentation.

LISTING 145: special1.tex using Listing 143

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

LISTING 146: special1.tex using Listing 144

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

We may specify `noAdditionalIndentGlobal` and `indentRulesGlobal` as in Listings 147 and 148.

LISTING 147: special-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
    specialBeginEnd: 1
```

LISTING 148: 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 149 and 150; notice that in Listing 149 neither of the `special` code blocks have received indentation, while in Listing 150 both code blocks have received a single space of indentation.

LISTING 149: special1.tex using Listing 147

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

LISTING 150: special1.tex using Listing 148

```
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 151 for demonstration throughout this Section. As discussed on Listing 69, by default `latexindent.pl` will not add indentation after headings.

LISTING 151: headings2.tex

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

On using the YAML file in Listing 153 by running the command

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

we obtain the output in Listing 152. Note that the argument of `paragraph` has received (default) indentation, and that the body after the heading statement has received (default) indentation.

LISTING 152: headings2.tex using Listing 153

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

LISTING 153: headings3.yaml

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

If we specify `noAdditionalIndent` as in Listing 155 and run the command

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

then we receive the output in Listing 154. 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 154: headings2.tex using Listing 155

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

LISTING 155: headings4.yaml

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

Similarly, if we specify `indentRules` as in Listing 157 and run analogous commands to those above, we receive the output in Listing 156; note that the *body*, *mandatory argument* and content *after the heading* of `paragraph` have *all* received three tabs worth of indentation.

LISTING 156: headings2.tex using Listing 157

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

LISTING 157: headings5.yaml

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

We may, instead, specify `noAdditionalIndent` in 'field' form, as in Listing 159 which gives the output in Listing 158.

LISTING 158: headings2.tex using Listing 159

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

LISTING 159: headings6.yaml

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

Analogously, we may specify `indentRules` as in Listing 161 which gives the output in Listing 160; 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 160: headings2.tex using Listing 161

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

LISTING 161: 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 163 and 165 respectively, with respective output in Listings 162 and 164. Note that in Listing 163 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 164, the *argument* has received both a (default) tab plus two spaces of indentation (from the global rule specified in Listing 165), and the remaining body after `paragraph` has received just two spaces of indentation.

LISTING 162: headings2.tex using Listing 163

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

LISTING 163: headings8.yaml

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

LISTING 164: headings2.tex using Listing 165

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

LISTING 165: 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 32, 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 40, 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 32;
- 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 166, with the default output given in Listing 167.

LISTING 166: pgfkeys1.tex

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

LISTING 167: pgfkeys1.tex default output

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

In Listing 167, 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 Section 5.2.

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

- it must immediately follow either *horizontal space* OR *one or more line breaks* OR `{` OR `[` OR `$` OR `)` OR `(`;
- the name may contain the characters detailed in Table 2 on page 32;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

A simple example is given in Listing 168, with default output in Listing 169.

LISTING 168: `child1.tex`

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

LISTING 169: `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`[8]. Referencing Listing 169, 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 Section 5.2.

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 170 with default output give in Listing 171.

LISTING 170: `psforeach1.tex`

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

LISTING 171: `psforeach1.tex` default output

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

Referencing Listing 171, 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;

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

- 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 Section 5.2.

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 172 and 173 should now make sense.

LISTING 172: noAdditionalIndentGlobal

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

LISTING 173: indentRulesGlobal

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

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*⟩

U: 2018-04-27

The `commandCodeBlocks` field contains a few switches detailed in Listing 174.

LISTING 174: commandCodeBlocks

```
commandCodeBlocks:
    roundParenthesesAllowed: 1
    stringsAllowedBetweenArguments:
      -
        amalgamate: 1
      - 'node'
      - 'at'
      - 'to'
      - 'decoration'
      - '\+\+'
      - '\-\-'
    commandNameSpecial:
      -
        amalgamate: 1
      - '@ifnextchar\['
```

```
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 175.

LISTING 175: pstricks1.tex

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

LISTING 176: 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 174, then latexindent.pl will allow round parenthesis between optional and mandatory arguments. In the case of the code in Listing 175, latexindent.pl finds *all* the arguments of defFunction, both before and after (u,v).

The default output from running latexindent.pl on Listing 175 actually leaves it unchanged (see Listing 176); note in particular, this is because of noAdditionalIndentGlobal as discussed on Section 5.2.3.

Upon using the YAML settings in Listing 178, and running the command

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

we obtain the output given in Listing 177.

LISTING 177: pstricks1.tex using Listing 178

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

LISTING 178: noRoundParentheses.yaml

```
commandCodeBlocks:
    roundParenthesesAllowed: 0
```

Notice the difference between Listing 176 and Listing 177; in particular, in Listing 177, 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 32) which, by default, assume indentation for their body, and hence the tabbed indentation in Listing 177.

Let's explore this using the YAML given in Listing 180 and run the command

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

then the output is as in Listing 179.

LISTING 179: pstricks1.tex using Listing 180

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

LISTING 180: defFunction.yaml

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

Notice in Listing 179 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 180.

`stringsAllowedBetweenArguments`: ⟨*fields*⟩

`tikz` users may well specify code such as that given in Listing 181; processing this code using `latexindent.pl` gives the default output in Listing 182.

LISTING 181: `tikz-node1.tex`

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

LISTING 182: `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 174 on page 49, we see that the strings

to, node, ++

are all allowed to appear between arguments; importantly, you are encouraged to add further names to this field as necessary. This means that when `latexindent.pl` processes Listing 181, 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 184 and running the command

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

we receive the output given in Listing 183.

LISTING 183: `tikz-node1.tex` using Listing 184

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

LISTING 184: `draw.yaml`

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

Notice that each line after the `\draw` command (its 'body') in Listing 183 has been given the appropriate two-spaces worth of indentation specified in Listing 184.

Let's compare this with the output from using the YAML settings in Listing 186, and running the command

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

given in Listing 185.

LISTING 185: tikz-node1.tex using Listing 186

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

LISTING 186: no-strings.yaml

```
commandCodeBlocks:
    stringsAllowedBetweenArguments: 0
```

In this case, latexindent.pl sees that:

- the \draw command finishes after the (c), as stringsAllowedBetweenArguments has been set to 0 so there are no strings allowed between arguments;
- it finds a namedGroupingBracesBrackets called to (see Table 2 on page 32) *with* argument [in=110,out=-90]
- it finds another namedGroupingBracesBrackets but this time called node with argument [below,align=left,scale=0.5]

U: 2018-04-27

Referencing Listing 174 on page 49,, we see that the first field in the stringsAllowedBetweenArguments is amalgamate and is set to 1 by default. This is for users who wish to specify their settings in multiple YAML files. For example, by using the settings in either Listing 187 orListing 188 is equivalent to using the settings in Listing 189.

LISTING 187: amalgamate-demo.yaml

```
commandCodeBlocks:

    stringsAllowedBetweenArguments:
      - 'more'
      - 'strings'
      - 'here'
```

LISTING 188: amalgamate-demo1.yaml

```
commandCodeBlocks:

    stringsAllowedBetweenArguments:
      -
        amalgamate: 1
      - 'more'
      - 'strings'
      - 'here'
```

LISTING 189: amalgamate-demo2.yaml

```
commandCodeBlocks:

    stringsAllowedBetweenArguments:
      -
        amalgamate: 1
      - 'node'
      - 'at'
      - 'to'
      - 'decoration'
      - '\+\+'
      - '\-\-'
      - 'more'
      - 'strings'
      - 'here'
```

We specify amalgamate to be set to 0 and in which case any settings loaded prior to those specified, including the default, will be overwritten. For example, using the settings in Listing 190 means that only the strings specified in that field will be used.

LISTING 190: amalgamate-demo3.yaml

```
commandCodeBlocks:
    stringsAllowedBetweenArguments:
      -
        amalgamate: 0
      - 'further'
      - 'settings'
```

It is important to note that the amalgamate field, if used, must be in the first field, and specified using the syntax given in Listings 188 to 190.

We may explore this feature further with the code in Listing 191, whose default output is given in Listing 192.

LISTING 191: for-each.tex

```
\foreach \x/\y in {0/1,1/2}{
body of foreach
}
```

LISTING 192: for-each default output

```
\foreach \x/\y in {0/1,1/2}{
        body of foreach
    }
```

Let's compare this with the output from using the YAML settings in Listing 194, and running the command

```
cmh:~$ latexindent.pl for-each.tex -l foreach.yaml
```

given in Listing 193.

LISTING 193: for-each.tex using Listing 194

```
\foreach \x/\y in {0/1,1/2}{
    body of foreach
}
```

LISTING 194: foreach.yaml

```
commandCodeBlocks:
    stringsAllowedBetweenArguments:
      -
        amalgamate: 0
      - '\\x\/\\y'
      - 'in'
```

You might like to compare the output given in Listing 192 and Listing 193. Note,in particular, in Listing 192 that the `foreach` command has not included any of the subsequent strings, and that the braces have been treated as a `namedGroupingBracesBrackets`. In Listing 193 the `foreach` command has been allowed to have `\x/\y` and `in` between arguments because of the settings given in Listing 194.

`commandNameSpecial`: ⟨*fields*⟩

U: 2018-04-27

There are some special command names that do not fit within the names recognized by `latexindent.pl`, the first one of which is `\@ifnextchar[`. From the perspective of `latexindent.pl`, the whole of the text `\@ifnextchar[` is a command, because it is immediately followed by sets of mandatory arguments. However, without the `commandNameSpecial` field, `latexindent.pl` would not be able to label it as such, because the `[` is, necessarily, not matched by a closing `]`.

For example, consider the sample file in Listing 195, which has default output in Listing 196.

LISTING 195: ifnextchar.tex

```
\parbox{
\@ifnextchar[{arg 1}{arg 2}
}
```

LISTING 196: ifnextchar.tex default output

```
\parbox{
    \@ifnextchar[{arg 1}{arg 2}
}
```

Notice that in Listing 196 the `parbox` command has been able to indent its body, because `latexindent.pl` has successfully found the command `\@ifnextchar` first; the pattern-matching of `latexindent.pl` starts from *the inner most <thing> and works outwards*, discussed in more detail on Section 6.8.

For demonstration, we can compare this output with that given in Listing 197 in which the settings from Listing 198 have dictated that no special command names, including the `\@ifnextchar[` command, should not be searched for specially; as such, the `parbox` command has been *unable* to indent its body successfully, because the `\@ifnextchar[` command has not been found.

LISTING 197: ifnextchar.tex using Listing 198

```
\parbox{
\@ifnextchar[{arg 1}{arg 2}
}
```

LISTING 198: no-ifnextchar.yaml

```
commandCodeBlocks:
    commandNameSpecial: 0
```

The `amalgamate` field can be used for `commandNameSpecial`, just as for `stringsAllowedBetweenArguments`. The same condition holds as stated previously, which we state again here:

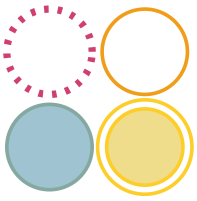
It is important to note that the `amalgamate` field, if used, in either `commandNameSpecial` or `stringsAllowedBetweenArguments` must be in the first field, and specified using the syntax given in Listings 188 to 190.

6 The -m (modifylinebreaks) switch

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

6.1 oneSentencePerLine: modifying line breaks for sentences 57
6.1.1 sentencesFollow 59
6.1.2 sentencesBeginWith 60
6.1.3 sentencesEndWith 61
6.1.4 Features of the oneSentencePerLine routine 63
6.2 removeParagraphLineBreaks: modifying line breaks for paragraphs 64
6.3 Poly-switches 70
6.4 modifyLineBreaks for environments 71
6.4.1 Adding line breaks: BeginStartsOnOwnLine and BodyStartsOnOwnLine 71
6.4.2 Adding line breaks using EndStartsOnOwnLine and EndFinishesWithLineBreak 72
6.4.3 poly-switches only add line breaks when necessary 73
6.4.4 Removing line breaks (poly-switches set to −1) 74
6.4.5 About trailing horizontal space 75
6.4.6 poly-switch line break removal and blank lines 75
6.5 Poly-switches for other code blocks 77

`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 199.

LISTING 199: `modifyLineBreaks` `-m`

```
modifyLineBreaks:
    preserveBlankLines: 1
    condenseMultipleBlankLinesInto: 1
```

Having read the previous paragraph, it should sound reasonable that, if you call `latexindent.pl` using the `-m` switch, then you give it permission to modify line breaks in your file, but let's be clear:

If you call `latexindent.pl` with the `-m` switch, then you are giving it permission to modify line breaks. By default, the only thing that will happen is that multiple blank lines will be condensed into one blank line; many other settings are possible, discussed next.

`preserveBlankLines`: **0|1**

This field is directly related to *poly-switches*, discussed below. By default, it is set to `1`, which means that blank lines will be protected from removal; however, regardless of this setting, multiple blank lines can be condensed if `condenseMultipleBlankLinesInto` is greater than `0`, discussed next.

`condenseMultipleBlankLinesInto`: ⟨*positive integer*⟩

Assuming that this switch takes an integer value greater than 0, `latexindent.pl` will condense multiple blank lines into the number of blank lines illustrated by this switch. As an example, Listing 200 shows a sample file with blank lines; upon running

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

the output is shown in Listing 201; note that the multiple blank lines have been condensed into one blank line, and note also that we have used the `-m` switch!

LISTING 200: `mlb1.tex`

```
before blank line



after blank line


after blank line
```

LISTING 201: `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 202. 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 202: `textWrapOptions` `-m`

```
397     textWrapOptions:
398         columns: 0
```

For example, consider the file give in Listing 203.

LISTING 203: `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 205, and running the command

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

we obtain the output in Listing 204.

LISTING 204: 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 205: textwrap1.yaml -m

```
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 206,

LISTING 206: 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 205,

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

then the output is as in Listing 207.

LISTING 207: 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 208

LISTING 208: 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 205,

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

then the output is as in Listing 209.

LISTING 209: `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 [15]). By default, the separator is empty (see Listing 210) which means that a new line token will be used, but you can change it as you see fit.

LISTING 210: `textWrapOptions` -m

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

For example starting with the file in Listing 211

LISTING 211: `textwrap4.tex`

```
Here is a line of text.
```

and using `textwrap2.yaml` from Listing 213 with the following command

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

then we obtain the output in Listing 212.

LISTING 212: `textwrap4-mod2.tex`

```
Here||is a||line||of||text||.
```

LISTING 213: `textwrap2.yaml` -m

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

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

- Verbatim environments (Listing 17 on page 19) and verbatim commands (Listing 18 on page 19) will *not* be affected by the text wrapping routine (see Listing 207 on the previous page);
- comments will *not* be affected by the text wrapping routine (see Listing 209);
- indentation is performed *after* the text wrapping routine; as such, indented code will likely exceed any maximum value set in the `columns` field.

6.1 oneSentencePerLine: modifying line breaks for sentences

N: 2018-01-13

You can instruct `latexindent.pl` to format your file so that it puts one sentence per line. Thank you to [10] for helping to shape and test this feature. The behaviour of this part of the script is controlled by the switches detailed in Listing 214, all of which we discuss next.

LISTING 214: oneSentencePerLine -m

```
400     oneSentencePerLine:
401         manipulateSentences: 0
402         removeSentenceLineBreaks: 1
403         sentencesFollow:
404             par: 1
405             blankLine: 1
406             fullStop: 1
407             exclamationMark: 1
408             questionMark: 1
409             rightBrace: 1
410             commentOnPreviousLine: 1
411             other: 0
412         sentencesBeginWith:
413             A-Z: 1
414             a-z: 0
415             other: 0
416         sentencesEndWith:
417             basicFullStop: 0
418             betterFullStop: 1
419             exclamationMark: 1
420             questionMark: 1
421             other: 0
```

manipulateSentences: 0|1

This is a binary switch that details if `latexindent.pl` should perform the sentence manipulation routine; it is *off* (set to 0) by default, and you will need to turn it on (by setting it to 1) if you want the script to modify line breaks surrounding and within sentences.

removeSentenceLineBreaks: 0|1

When operating upon sentences `latexindent.pl` will, by default, remove internal linebreaks as `removeSentenceLineBreaks` is set to 1. Setting this switch to 0 instructs `latexindent.pl` not to do so.

For example, consider `multiple-sentences.tex` shown in Listing 215.

LISTING 215: multiple-sentences.tex

```
This is the first
sentence. This is the; second, sentence. This is the
third sentence.

This is the fourth
sentence! This is the fifth sentence? This is the
sixth sentence.
```

If we use the YAML files in Listings 217 and 219, and run the commands

```
cmh:~$ latexindent.pl multiple-sentences -m -l=manipulate-sentences.yaml
cmh:~$ latexindent.pl multiple-sentences -m -l=keep-sen-line-breaks.yaml
```

then we obtain the respective output given in Listings 216 and 218.

LISTING 216: multiple-sentences.tex using Listing 217

```
This is the first sentence.
This is the; second, sentence.
This is the third sentence.

This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 217: manipulate-sentences.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
```

LISTING 218: multiple-sentences.tex using Listing 219

```
This is the first
sentence.
This is the; second, sentence.
This is the
third sentence.

This is the fourth
sentence!
This is the fifth sentence?
This is the
sixth sentence.
```

LISTING 219: keep-sen-line-breaks.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        removeSentenceLineBreaks: 0
```

Notice, in particular, that the 'internal' sentence line breaks in Listing 215 have been removed in Listing 216, but have not been removed in Listing 218.

The remainder of the settings displayed in Listing 214 on the previous page instruct `latexindent.pl` on how to define a sentence. From the perpesctive of `latexindent.pl` a sentence must:

- *follow* a certain character or set of characters (see Listing 220); by default, this is either `\par`, a blank line, a full stop/period (.), exclamation mark (!), question mark (?) right brace (}) or a comment on the previous line;
- *begin* with a character type (see Listing 221); by default, this is only capital letters;
- *end* with a character (see Listing 222); by default, these are full stop/period (.), exclamation mark (!) and question mark (?).

In each case, you can specify the `other` field to include any pattern that you would like; you can specify anything in this field using the language of regular expressions.

LISTING 220: sentencesFollow -m

```
sentencesFollow:
    par: 1
    blankLine: 1
    fullStop: 1
    exclamationMark: 1
    questionMark: 1
    rightBrace: 1
    commentOnPreviousLine: 1
    other: 0
```

LISTING 221: sentencesBeginWith -m

```
sentencesBeginWith:
    A-Z: 1
    a-z: 0
    other: 0
```

LISTING 222: sentencesEndWith -m

```
sentencesEndWith:
    basicFullStop: 0
    betterFullStop: 1
    exclamationMark: 1
    questionMark: 1
    other: 0
```

6.1.1 sentencesFollow

Let's explore a few of the switches in `sentencesFollow`; let's start with Listing 215 on the preceding page, and use the YAML settings given in Listing 224. Using the command

```
cmh:~$ latexindent.pl multiple-sentences -m -l=sentences-follow1.yaml
```

we obtain the output given in Listing 223.

LISTING 223: multiple-sentences.tex using Listing 224

```
This is the first sentence.
This is the; second, sentence.
This is the third sentence.

This is the fourth
sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 224: sentences-follow1.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesFollow:
            blankLine: 0
```

Notice that, because `blankLine` is set to `0`, `latexindent.pl` will not seek sentences following a blank line, and so the fourth sentence has not been accounted for.

We can explore the `other` field in Listing 220 with the `.tex` file detailed in Listing 225.

LISTING 225: multiple-sentences1.tex

```
(Some sentences stand alone in brackets.) This is the first
sentence. This is the; second, sentence. This is the
third sentence.
```

Upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences1 -m -l=manipulate-sentences.yaml
cmh:~$ latexindent.pl multiple-sentences1 -m -l=manipulate-sentences.yaml,sentences-follow2.yaml
```

then we obtain the respective output given in Listings 226 and 227.

LISTING 226: multiple-sentences1.tex using Listing 217 on the previous page

```
(Some sentences stand alone in brackets.) This is the first
sentence.
This is the; second, sentence.
This is the third sentence.
```

LISTING 227: multiple-sentences1.tex using Listing 228

```
(Some sentences stand alone in brackets.)
This is the first sentence.
This is the; second, sentence.
This is the third sentence.
```

LISTING 228: sentences-follow2.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesFollow:
            other: "\)"
```

Notice that in Listing 226 the first sentence after the) has not been accounted for, but that following the inclusion of Listing 228, the output given in Listing 227 demonstrates that the sentence *has* been accounted for correctly.

6.1.2 sentencesBeginWith

By default, `latexindent.pl` will only assume that sentences begin with the upper case letters `A-Z`; you can instruct the script to define sentences to begin with lower case letters (see Listing 221), and we can use the `other` field to define sentences to begin with other characters.

LISTING 229: multiple-sentences2.tex

```
This is the first
sentence.

$a$ can
represent a
number. 7 is
at the beginning of this sentence.
```

Upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences2 -m -l=manipulate-sentences.yaml
cmh:~$ latexindent.pl multiple-sentences2 -m -l=manipulate-sentences.yaml,sentences-begin1.yaml
```

then we obtain the respective output given in Listings 230 and 231.

LISTING 230: multiple-sentences2.tex using Listing 217 on page 59

```
This is the first sentence.

$a$ can
represent a
number. 7 is
at the beginning of this sentence.
```

LISTING 231: multiple-sentences2.tex using Listing 232

```
This is the first sentence.

$a$ can represent a number.
7 is at the beginning of this sentence.
```

LISTING 232: sentences-begin1.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesBeginWith:
            other: "\$|[0-9]"
```

Notice that in Listing 230, the first sentence has been accounted for but that the subsequent sentences have not. In Listing 231, all of the sentences have been accounted for, because the `other` field in Listing 232 has defined sentences to begin with either `$` or any numeric digit, `0` to `9`.

6.1.3 sentencesEndWith

Let's return to Listing 215 on page 58; we have already seen the default way in which `latexindent.pl` will operate on the sentences in this file in Listing 216 on page 59. We can populate the `other` field with any character that we wish; for example, using the YAML specified in Listing 234 and the command

```
cmh:~$ latexindent.pl multiple-sentences -m -l=sentences-end1.yaml
cmh:~$ latexindent.pl multiple-sentences -m -l=sentences-end2.yaml
```

then we obtain the output in Listing 233.

LISTING 233: multiple-sentences.tex using Listing 234

```
This is the first sentence.
This is the;
second, sentence.
This is the third sentence.

This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 234: sentences-end1.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesEndWith:
          other: "\:|\;|\,"
```

LISTING 235: multiple-sentences.tex using Listing 236

```
This is the first sentence.
This is the;
second,
sentence.
This is the third sentence.

This is the fourth sentence!
This is the fifth sentence?
This is the sixth sentence.
```

LISTING 236: sentences-end2.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesEndWith:
          other: "\:|\;|\,"
        sentencesBeginWith:
          a-z: 1
```

There is a subtle difference between the output in Listings 233 and 235; in particular, in Listing 233 the word `sentence` has not been defined as a sentence, because we have not instructed `latexindent.pl` to begin sentences with lower case letters. We have changed this by using the settings in Listing 236, and the associated output in Listing 235 reflects this.

Referencing Listing 222 on page 59, you'll notice that there is a field called `basicFullStop`, which is set to 0, and that the `betterFullStop` is set to 1 by default.

Let's consider the file shown in Listing 237.

LISTING 237: url.tex

```
This sentence, \url{tex.stackexchange.com/} finishes here. Second sentence.
```

Upon running the following commands

```
cmh:~$ latexindent.pl url -m -l=manipulate-sentences.yaml
```

we obtain the output given in Listing 238.

LISTING 238: url.tex using Listing 217 on page 59

```
This sentence, \url{tex.stackexchange.com/} finishes here.
Second sentence.
```

Notice that the full stop within the url has been interpreted correctly. This is because, within the `betterFullStop`, full stops at the end of sentences have the following properties:

- they are ignored within `e.g.` and `i.e.`;
- they can not be immediately followed by a lower case or upper case letter;
- they can not be immediately followed by a hyphen, comma, or number.

If you find that the `betterFullStop` does not work for your purposes, then you can switch it off by setting it to 0, and you can experiment with the `other` field.

The `basicFullStop` routine should probably be avoided in most situations, as it does not accomodate the specifications above. For example, using the following command

```
cmh:~$ latexindent.pl url -m -l=alt-full-stop1.yaml
```

and the YAML in Listing 240 gives the output in Listing 239.

LISTING 239: url.tex using Listing 240

```
This sentence, \url{tex.
    stackexchange.com/} finishes here.Second sentence.
```

LISTING 240: alt-full-stop1.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesEndWith:
            basicFullStop: 1
            betterFullStop: 0
```

Notice that the full stop within the URL has not been accomodated correctly because of the non-default settings in Listing 240.

6.1.4 Features of the oneSentencePerLine routine

The sentence manipulation routine takes place *after* verbatim environments, preamble and trailing comments have been accounted for; this means that any characters within these types of code blocks will not be part of the sentence manipulation routine.

For example, if we begin with the .tex file in Listing 241, and run the command

```
cmh:~$ latexindent.pl multiple-sentences3 -m -l=manipulate-sentences.yaml
```

then we obtain the output in Listing 242.

LISTING 241: multiple-sentences3.tex

```
The first sentence continues after the verbatim
\begin{verbatim}
  there are sentences within this. These
  will not be operated
  upon by latexindent.pl.
\end{verbatim}
and finishes here. Second sentence % a commented full stop.
contains trailing comments,
which are ignored.
```

LISTING 242: multiple-sentences3.tex using Listing 217 on page 59

```
The first sentence continues after the verbatim \begin{verbatim}
  there are sentences within this. These
  will not be operated
  upon by latexindent.pl.
\end{verbatim} and finishes here.
Second sentence contains trailing comments, which are ignored.
% a commented full stop.
```

Furthermore, if sentences run across environments then, by default, the line breaks internal to the sentence will be removed. For example, if we use the .tex file in Listing 243 and run the commands

```
cmh:~$ latexindent.pl multiple-sentences4 -m -l=manipulate-sentences.yaml
cmh:~$ latexindent.pl multiple-sentences4 -m -l=keep-sen-line-breaks.yaml
```

then we obtain the output in Listings 244 and 245.

LISTING 243: multiple-sentences4.tex

```
This sentence
\begin{itemize}
  \item continues
\end{itemize}
across itemize
and finishes here.
```

LISTING 244: multiple-sentences4.tex using Listing 217 on page 59

```
This sentence \begin{itemize} \item continues \end{itemize} across itemize and finishes here.
```

LISTING 245: multiple-sentences4.tex using Listing 219 on page 59

```
This sentence
\begin{itemize}
    \item continues
\end{itemize}
across itemize
and finishes here.
```

Once you've read Section 6.3, you will know that you can accomodate the removal of internal sentence line breaks by using the YAML in Listing 247 and the command

```
cmh:~$ latexindent.pl multiple-sentences4 -m -l=item-rules2.yaml
```

the output of which is shown in Listing 246.

LISTING 246: multiple-sentences4.tex using Listing 247

```
This sentence
\begin{itemize}
    \item continues
\end{itemize}
across itemize and finishes here.
```

LISTING 247: item-rules2.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
    items:
        ItemStartsOnOwnLine: 1
    environments:
        BeginStartsOnOwnLine: 1
        BodyStartsOnOwnLine: 1
        EndStartsOnOwnLine: 1
        EndFinishesWithLineBreak: 1
```

6.2 removeParagraphLineBreaks: modifying line breaks for paragraphs

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 248. Thank you to [11] for shaping and assisting with the testing of this feature.

removeParagraphLineBreaks: ⟨*fields*⟩

This feature is considered complimentary to the oneSentencePerLine feature described in Section 6.1 on page 57.

LISTING 248: removeParagraphLineBreaks -m

```
422     removeParagraphLineBreaks:
423         all: 0
424         alignAtAmpersandTakesPriority: 1
425         environments:
426             quotation: 0
427         ifElseFi: 0
428         optionalArguments: 0
429         mandatoryArguments: 0
430         items: 0
431         specialBeginEnd: 0
432         afterHeading: 0
433         filecontents: 0
434         masterDocument: 0
```

This routine can be turned on *globally* for *every* code block type known to `latexindent.pl` (see Table 2 on page 32) 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 32;
- 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 265 on page 69.

Let's start with the `.tex` file in Listing 249, together with the YAML settings in Listing 250.

LISTING 249: shortlines.tex

```
\begin{myenv}
The␣lines
in␣this
environment
are␣very
short
and␣contain
many␣linebreaks.

Another
paragraph.
\end{myenv}
```

LISTING 250: remove-para1.yaml -m

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

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 251.

LISTING 251: 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 249 which has crept in to the output in Listing 251. This can be fixed using the YAML file in Listing 312

on page 75 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 252; notice that the double spaces present in Listing 251 have been addressed.

LISTING 252: 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 250, we note that the `all` switch applies to *all* code block types. So, for example, let's consider the files in Listings 253 and 254

LISTING 253: shortlines-mand.tex

```
\mycommand{
The lines
in this
command
are very
short
and contain
many linebreaks.

Another
paragraph.
}
```

LISTING 254: 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 255 and 256.

LISTING 255: shortlines-mand1.tex

```
\mycommand{
    The lines in this  command are very  short and contain many linebreaks.

    Another  paragraph.
}
```

LISTING 256: 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 257, and consider the settings in Listings 258 and 259; note that in Listing 258 we specify that *every* environment should receive treatment from the routine,

while in Listing 259 we specify that *only* the one environment should receive the treatment.

LISTING 257: 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}
```

LISTING 258: remove-para2.yaml -m

```
modifyLineBreaks:
    removeParagraphLineBreaks:
        environments: 1
```

LISTING 259: remove-para3.yaml -m

```
modifyLineBreaks:
    removeParagraphLineBreaks:
        environments:
            one: 1
```

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 260 and 261.

LISTING 260: 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 261: 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 262, with the YAML settings in Listing 263.

LISTING 262: 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 263: 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 264.

LISTING 264: 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 265.

LISTING 265: paragraphsStopAt -m

```
435         paragraphsStopAt:
436             environments: 1
437             commands: 0
438             ifElseFi: 0
439             items: 0
440             specialBeginEnd: 0
441             heading: 0
442             filecontents: 0
443             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 265. 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 266; we will, in turn, consider the settings in Listings 267 and 268.

LISTING 266: 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 267: stop-command.yaml -m

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

LISTING 268: stop-comment.yaml -m

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

Upon using the settings from Listing 263 on the previous page 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 269 to 271; notice in particular that:

- in Listing 269 the paragraph line break routine has included commands and comments;
- in Listing 270 the paragraph line break routine has *stopped* at the `emph` command, because in Listing 267 we have specified `commands` to be `1`, and `emph` is at the beginning of a line;
- in Listing 271 the paragraph line break routine has *stopped* at the comments, because in Listing 268 we have specified `comments` to be `1`, and the comment is at the beginning of a line.

In all outputs in Listings 269 to 271 we notice that the paragraph line break routine has stopped at `\begin{myenv}` because, by default, `environments` is set to `1` in Listing 265 on the preceding page.

LISTING 269: 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 270: 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 271: 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
```

6.3 Poly-switches

U: 2017-08-21

Every other field in the `modifyLineBreaks` field uses poly-switches, and can take one of *five* integer values:

−1 *remove mode*: line breaks before or after the *<part of thing>* can be removed (assuming that `preserveBlankLines` is set to `0`);

0 *off mode*: line breaks will not be modified for the *<part of thing>* under consideration;

1 *add mode*: a line break will be added before or after the *<part of thing>* under consideration, assuming that there is not already a line break before or after the *<part of thing>*;

2 *comment then add mode*: a comment symbol will be added, followed by a line break before or after the *<part of thing>* under consideration, assuming that there is not already a comment and line break before or after the *<part of thing>*;

N: 2017-08-21

3 *add then blank line mode*: a line break will be added before or after the *<part of thing>* under consideration, assuming that there is not already a line break before or after the *<part of thing>*, followed by a blank line.

In the above, *<part of thing>* refers to either the *begin statement*, *body* or *end statement* of the code blocks detailed in Table 2 on page 32. 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.

6.4 modifyLineBreaks for environments

We start by viewing a snippet of `defaultSettings.yaml` in Listing 272; note that it contains *global* settings (immediately after the `environments` field) and that *per-name* settings are also allowed – in the case of Listing 272, settings for `equation*` have been specified. Note that all poly-switches are *off* by default.

LISTING 272: environments -m

```
    environments:
        BeginStartsOnOwnLine: 0
        BodyStartsOnOwnLine: 0
        EndStartsOnOwnLine: 0
        EndFinishesWithLineBreak: 0
        equation*:
            BeginStartsOnOwnLine: 0
            BodyStartsOnOwnLine: 0
            EndStartsOnOwnLine: 0
            EndFinishesWithLineBreak: 0
```

Let's begin with the simple example given in Listing 273; note that we have annotated key parts of the file using ♠, ♡, ♢ and ♣, these will be related to fields specified in Listing 272.

LISTING 273: env-mlb1.tex

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

6.4.1 Adding line breaks: BeginStartsOnOwnLine and BodyStartsOnOwnLine

Let's explore `BeginStartsOnOwnLine` and `BodyStartsOnOwnLine` in Listings 274 and 275, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 274: env-mlb1.yaml -m

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

LISTING 275: 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 276 and 277 respectively.

LISTING 276: env-mlb.tex using Listing 274

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

LISTING 277: env-mlb.tex using Listing 275

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

There are a couple of points to note:

- in Listing 276 a line break has been added at the point denoted by ♠ in Listing 273; no other line breaks have been changed;
- in Listing 277 a line break has been added at the point denoted by ♡ in Listing 273; furthermore, note that the *body* of `myenv` has received the appropriate (default) indentation.

Let's now change each of the 1 values in Listings 274 and 275 so that they are 2 and save them into `env-mlb3.yaml` and `env-mlb4.yaml` respectively (see Listings 278 and 279).

LISTING 278: `env-mlb3.yaml` -m

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

LISTING 279: `env-mlb4.yaml` -m

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

Upon running commands analogous to the above, we obtain Listings 280 and 281.

LISTING 280: `env-mlb.tex` using Listing 278

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

LISTING 281: `env-mlb.tex` using Listing 279

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

Note that line breaks have been added as in Listings 276 and 277, 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.

N: 2017-08-21

Let's now change each of the 1 values in Listings 274 and 275 so that they are 3 and save them into `env-mlb5.yaml` and `env-mlb6.yaml` respectively (see Listings 282 and 283).

LISTING 282: `env-mlb5.yaml` -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 3
```

LISTING 283: `env-mlb6.yaml` -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 3
```

Upon running commands analogous to the above, we obtain Listings 284 and 285.

LISTING 284: `env-mlb.tex` using Listing 282

```
before words

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

LISTING 285: `env-mlb.tex` using Listing 283

```
before words \begin{myenv}

    body of myenv\end{myenv} after words
```

Note that line breaks have been added as in Listings 276 and 277, but this time a *blank line* has been added after adding the line break.

6.4.2 Adding line breaks using EndStartsOnOwnLine and EndFinishesWithLineBreak

Let's explore `EndStartsOnOwnLine` and `EndFinishesWithLineBreak` in Listings 286 and 287, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 286: `env-mlb7.yaml` -m

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

LISTING 287: `env-mlb8.yaml` -m

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

After running the following commands,

```
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb7.yaml
cmh:~$ latexindent.pl -m env-mlb.tex -l env-mlb8.yaml
```

the output is as in Listings 288 and 289.

LISTING 288: `env-mlb.tex` using Listing 286

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

LISTING 289: `env-mlb.tex` using Listing 287

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

There are a couple of points to note:

- in Listing 288 a line break has been added at the point denoted by ♢ in Listing 273 on page 71; no other line breaks have been changed and the `\end{myenv}` statement has *not* received indentation (as intended);
- in Listing 289 a line break has been added at the point denoted by ♣ in Listing 273 on page 71.

Let's now change each of the 1 values in Listings 286 and 287 so that they are 2 and save them into `env-mlb9.yaml` and `env-mlb10.yaml` respectively (see Listings 290 and 291).

LISTING 290: `env-mlb9.yaml` -m

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

LISTING 291: `env-mlb10.yaml` -m

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

Upon running commands analogous to the above, we obtain Listings 292 and 293.

LISTING 292: `env-mlb.tex` using Listing 290

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

LISTING 293: `env-mlb.tex` using Listing 291

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

Note that line breaks have been added as in Listings 288 and 289, 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.

N: 2017-08-21

Let's now change each of the 1 values in Listings 286 and 287 so that they are 3 and save them into `env-mlb11.yaml` and `env-mlb12.yaml` respectively (see Listings 294 and 295).

LISTING 294: `env-mlb11.yaml` -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 3
```

LISTING 295: `env-mlb12.yaml` -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 3
```

Upon running commands analogous to the above, we obtain Listings 296 and 297.

LISTING 296: `env-mlb.tex` using Listing 294

```
before words \begin{myenv}body of myenv

\end{myenv} after words
```

LISTING 297: `env-mlb.tex` using Listing 295

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

after words
```

Note that line breaks have been added as in Listings 288 and 289, and that a *blank line* has been added after the line break.

6.4.3 poly-switches only add line breaks when necessary

If you ask `latexindent.pl` to add a line break (possibly with a comment) using a poly-switch value of 1 (or 2), it will only do so if necessary. For example, if you process the file in Listing 298 using any of the YAML files presented so far in this section, it will be left unchanged.

LISTING 298: `env-mlb2.tex`

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

LISTING 299: `env-mlb3.tex`

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

In contrast, the output from processing the file in Listing 299 will vary depending on the poly-switches used; in Listing 300 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 301 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 299 and by setting the other poly-switches considered so far to 2 in turn.

LISTING 300: env-mlb3.tex using Listing 275 on page 71

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

LISTING 301: env-mlb3.tex using Listing 279 on page 72

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

The details of the discussion in this section have concerned *global* poly-switches in the `environments` field; each switch can also be specified on a *per-name* basis, which would take priority over the global values; with reference to Listing 272 on page 71, an example is shown for the `equation*` environment.

6.4.4 Removing line breaks (poly-switches set to −1)

Setting poly-switches to −1 tells `latexindent.pl` to remove line breaks of the *<part of the thing>*, if necessary. We will consider the example code given in Listing 302, noting in particular the positions of the line break highlighters, ♠, ♡, ♢ and ♣, together with the associated YAML files in Listings 303 to 306.

LISTING 302: env-mlb4.tex

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

LISTING 303: env-mlb13.yaml -m

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

LISTING 304: env-mlb14.yaml -m

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

LISTING 305: env-mlb15.yaml -m

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

LISTING 306: env-mlb16.yaml -m

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

After running the commands

```
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb13.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb14.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb15.yaml
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb16.yaml
```

we obtain the respective output in Listings 307 to 310.

LISTING 307: env-mlb4.tex using Listing 303

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

LISTING 308: env-mlb4.tex using Listing 304

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

LISTING 309: env-mlb4.tex using Listing 305

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

LISTING 310: env-mlb4.tex using Listing 306

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

Notice that in:

- Listing 307 the line break denoted by ♠ in Listing 302 has been removed;
- Listing 308 the line break denoted by ♡ in Listing 302 has been removed;
- Listing 309 the line break denoted by ♢ in Listing 302 has been removed;
- Listing 310 the line break denoted by ♣ in Listing 302 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 303 to 306 into one file; alternatively, you could tell `latexindent.pl` to load them all by using the following command, for example

```
cmh:~$ latexindent.pl -m env-mlb4.tex -l env-mlb13.yaml,env-mlb14.yaml,env-mlb15.yaml,env-mlb16.yaml
```

which gives the output in Listing 273 on page 71.

6.4.5 About trailing horizontal space

Recall that on Section 5 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 311, which highlights trailing spaces.

LISTING 311: env-mlb5.tex

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

LISTING 312: removeTWS-before.yaml

```
removeTrailingWhitespace:
    beforeProcessing: 1
```

The output from the following commands

```
cmh:~$ latexindent.pl -m env-mlb5.tex -l env-mlb13.yaml,env-mlb14.yaml,env-mlb15.yaml,env-mlb16.yaml
cmh:~$ latexindent.pl -m env-mlb5.tex -l
    env-mlb13.yaml,env-mlb14.yaml,env-mlb15.yaml,env-mlb16.yaml,removeTWS-before.yaml
```

is shown, respectively, in Listings 313 and 314; note that the trailing horizontal white space has been preserved (by default) in Listing 313, while in Listing 314, it has been removed using the switch specified in Listing 312.

LISTING 313: env-mlb5.tex using Listings 307 to 310

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

LISTING 314: env-mlb5.tex using Listings 307 to 310 *and* Listing 312

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

6.4.6 poly-switch line break removal and blank lines

Now let's consider the file in Listing 315, which contains blank lines.

LISTING 315: env-mlb6.tex

```
before words♠

\begin{myenv}♡

body of myenv♢

\end{myenv}♣

after words
```

LISTING 316: UnpreserveBlankLines.yaml -m

```
modifyLineBreaks:
    preserveBlankLines: 0
```

Upon running the following commands

```
cmh:~$ latexindent.pl -m env-mlb6.tex -l env-mlb13.yaml,env-mlb14.yaml,env-mlb15.yaml,env-mlb16.yaml
cmh:~$ latexindent.pl -m env-mlb6.tex -l
    env-mlb13.yaml,env-mlb14.yaml,env-mlb15.yaml,env-mlb16.yaml,UnpreserveBlankLines.yaml
```

we receive the respective outputs in Listings 317 and 318. In Listing 317 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 318, we have allowed the poly-switches to remove blank lines because, in Listing 316, we have set `preserveBlankLines` to 0.

LISTING 317: env-mlb6.tex using Listings 307 to 310

```
before words

\begin{myenv}

    body of myenv

\end{myenv}

after words
```

LISTING 318: env-mlb6.tex using Listings 307 to 310 *and* Listing 316

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

We can explore this further using the blank-line poly-switch value of 3; let's use the file given in Listing 319.

LISTING 319: env-mlb7.tex

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

Upon running the following commands

```
cmh:~$ latexindent.pl -m env-mlb7.tex -l env-mlb12.yaml,env-mlb13.yaml
cmh:~$ latexindent.pl -m env-mlb7.tex -l
    env-mlb13.yaml,env-mlb14.yaml,UnpreserveBlankLines.yaml
```

we receive the outputs given in Listings 320 and 321.

LISTING 320: env-mlb7-preserve.tex

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

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

LISTING 321: env-mlb7-no-preserve.tex

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

Notice that in:

- Listing 320 that `\end{one}` has added a blank line, because of the value of `EndFinishesWithLineBreak` in Listing 295 on page 73, and even though the line break ahead of `\begin{two}` should have been removed (because of `BeginStartsOnOwnLine` in Listing 303 on page 74), the blank line has been preserved by default;
- Listing 321, by contrast, has had the additional line-break removed, because of the settings in Listing 316.

6.5 Poly-switches for other code blocks

Rather than repeat the examples shown for the environment code blocks (in Section 6.4 on page 71), 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 48 on page 26) can *not* be modified using `latexindent.pl`.

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

Code block	Sample		Poly-switch mapping
environment	`before words`♠ `\begin{myenv}`♡ `body of myenv`◇ `\end{myenv}`♣ `after words`	♠ ♡ ◇ ♣	BeginStartsOnOwnLine BodyStartsOnOwnLine EndStartsOnOwnLine EndFinishesWithLineBreak
ifelsefi	`before words`♠ `\if...`♡ `body of if/or statement`▲ `\or`▼ `body of if/or statement`★ `\else`□ `body of else statement`◇ `\fi`♣ `after words`	♠ ♡ ▲ ▼ ★ □ ◇ ♣	IfStartsOnOwnLine BodyStartsOnOwnLine OrStartsOnOwnLine OrFinishesWithLineBreak ElseStartsOnOwnLine ElseFinishesWithLineBreak FiStartsOnOwnLine FiFinishesWithLineBreak
optionalArguments	`...`♠ `[`♡ `body of opt arg`◇ `]`♣ `...`	♠ ♡ ◇ ♣	LSqBStartsOnOwnLine[9] OptArgBodyStartsOnOwnLine RSqBStartsOnOwnLine RSqBFinishesWithLineBreak
mandatoryArguments	`...`♠ `{`♡ `body of mand arg`◇ `}`♣ `...`	♠ ♡ ◇ ♣	LCuBStartsOnOwnLine[10] MandArgBodyStartsOnOwnLine RCuBStartsOnOwnLine RCuBFinishesWithLineBreak
commands	`before words`♠ `\mycommand`♡ ⟨*arguments*⟩	♠ ♡	CommandStartsOnOwnLine CommandNameFinishesWithLineBreak
namedGroupingBraces Brackets	before words♠ myname♡ ⟨*braces/brackets*⟩	♠ ♡	NameStartsOnOwnLine NameFinishesWithLineBreak

N: 2018-04-27

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

keyEqualsValuesBracesBrackets	before words♠ key•=♡ ⟨*braces/brackets*⟩	♠ • ♡	KeyStartsOnOwnLine EqualsStartsOnOwnLine EqualsFinishesWithLineBreak
items	before words♠ \item♡ ...	♠ ♡	ItemStartsOnOwnLine ItemFinishesWithLineBreak
specialBeginEnd	before words♠ \[♡ body of special/middle★ \middle□ body of special/middle ♢ \]♣ after words	♠ ♡ ★ □ ♢ ♣	SpecialBeginStartsOnOwnLine SpecialBodyStartsOnOwnLine SpecialMiddleStartsOnOwnLine SpecialMiddleFinishesWithLineBreak SpecialEndStartsOnOwnLine SpecialEndFinishesWithLineBreak

N: 2018-04-27

6.6 Partnering BodyStartsOnOwnLine with argument-based poly-switches

Some poly-switches need to be partnered together; in particular, when line breaks involving the *first* argument of a code block need to be accounted for using both `BodyStartsOnOwnLine` (or its equivalent, see Table 3 on the previous page) and `LCuBStartsOnOwnLine` for mandatory arguments, and `LSqBStartsOnOwnLine` for optional arguments.

Let's begin with the code in Listing 322 and the YAML settings in Listing 324; with reference to Table 3 on the preceding page, the key `CommandNameFinishesWithLineBreak` is an alias for `BodyStartsOnOwnLine`.

LISTING 322: `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 323; note that the *second* mandatory argument beginning brace `{` has had its leading line break removed, but that the *first* brace has not.

LISTING 323: `mycommand1.tex` using Listing 324

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

LISTING 324: `mycom-mlb1.yaml` -m

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

Now let's change the YAML file so that it is as in Listing 326; upon running the analogous command to that given above, we obtain Listing 325; both beginning braces `{` have had their leading line breaks removed.

LISTING 325: mycommand1.tex using Listing 326

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

LISTING 326: mycom-mlb2.yaml -m

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

Now let's change the YAML file so that it is as in Listing 328; upon running the analogous command to that given above, we obtain Listing 327.

LISTING 327: mycommand1.tex using Listing 328

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

LISTING 328: mycom-mlb3.yaml -m

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

6.7 Conflicting poly-switches: sequential code blocks

It is very easy to have conflicting poly-switches; if we use the example from Listing 322 on the previous page, and consider the YAML settings given in Listing 330. The output from running

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

is given in Listing 330.

LISTING 329: mycommand1.tex using Listing 330

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

LISTING 330: mycom-mlb4.yaml -m

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

Studying Listing 330, 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 329, 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 332; upon running the command

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

we obtain the output given in Listing 331.

LISTING 331: mycommand1.tex using Listing 332

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

LISTING 332: 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 334, which give associated output in Listing 333.

LISTING 333: mycommand1.tex using Listing 334

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

LISTING 334: 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.

6.8 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 335, noting that it contains nested environments.

LISTING 335: nested-env.tex

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

Let's use the YAML settings given in Listing 337, which upon running the command

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

gives the output in Listing 336.

LISTING 336: nested-env.tex using Listing 337

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

LISTING 337: nested-env-mlb1.yaml -m

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

In Listing 336, 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 335, 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 336, 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 339; upon running the command

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

we obtain the output given in Listing 338.

LISTING 338: nested-env.tex using Listing 339

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

LISTING 339: nested-env-mlb2.yaml -m

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

During Phase 1:

- the two environment is found first, and the line break ahead of the \end{two} statement is not changed because EndStartsOnOwnLine is set to 1. Importantly, because, *at this stage*, \end{two} *does* finish with a line break, EndFinishesWithLineBreak causes no action.
- next, the one environment is found; the line break ahead of \end{one} is already present, and no action is needed.

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

7 Conclusions and known limitations

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

The main limitation is to do with the alignment routine discussed on Section 5; for example, consider the file given in Listing 340.

LISTING 340: matrix2.tex

```
\matrix (A){
c01 & c02 & c03 & c0q \\
c_{11} & c12 & \ldots & c1q \\
};
```

The default output is given in Listing 341, and it is clear that the alignment routine has not worked as hoped, but it is *expected*.

LISTING 341: matrix2.tex default output

```
\matrix (A){
    c01                             & c02 & c03    & c0q \\
    c_{11} & c12 & \ldots & c1q \\
};
```

The reason for the problem is that when latexindent.pl stores its code blocks (see Table 2 on page 32) it uses replacement tokens. The alignment routine is using the *length of the replacement token* in its measuring – I hope to be able to address this in the future.

There are other limitations to do with the multicolumn alignment routine (see Listing 39 on page 24); in particular, when working with codeblocks in which multicolumn commands overlap, the algorithm can fail.

Another limitation is to do with efficiency, particularly when the -m switch is active, as this adds many checks and processes. The current implementation relies upon finding and storing *every* code block (see the discussion on Section 6.8); it is hoped that, in a future version, only *nested* code blocks will need to be stored in the 'packing' phase, and that this will improve the efficiency of the script.

You can run latexindent on .sty, .cls and any file types that you specify in fileExtensionPreference (see Listing 15 on page 18); 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 Listing 20).

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] *Log4perl Perl module*. URL: http://search.cpan.org/~mschilli/Log-Log4perl-1.49/lib/Log/Log4perl.pm (visited on 09/24/2017).

[12] *Perlbrew*. URL: http://perlbrew.pl/ (visited on 01/23/2017).

[13] *Strawberry Perl*. URL: http://strawberryperl.com/ (visited on 01/23/2017).
[14] *Text::Tabs Perl module*. URL: http://search.cpan.org/~muir/Text-Tabs+Wrap-2013.0523/lib.old/Text/Tabs.pm (visited on 07/06/2017).
[15] *Text::Wrap Perl module*. URL: http://perldoc.perl.org/Text/Wrap.html (visited on 05/01/2017).
[16] *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] Harish Kumar. *Early version testing*. Nov. 10, 2013. URL: https://github.com/harishkumarholla (visited on 06/30/2017).
[10] mlep. *One sentence per line*. Aug. 16, 2017. URL: https://github.com/cmhughes/latexindent.pl/issues/81 (visited on 01/08/2018).
[11] John Owens. *Paragraph line break routine removal*. May 27, 2017. URL: https://github.com/cmhughes/latexindent.pl/issues/33 (visited on 05/27/2017).
[17] 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 342 (`perl helloworld.pl`) then you will be able to run `latexindent.pl`, otherwise you may need to install the missing modules – see appendices A.1 and A.2.

LISTING 342: `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 Text::Tabs;
use FindBin;
use YAML::Tiny;
use File::Copy;
use File::Basename;
use File::HomeDir;
use Getopt::Long;
use Data::Dumper;
use List::Util qw(max);
use Log::Log4perl qw(get_logger :levels);

print "hello world";
exit;
```

N: 2018-01-13

A.1 Module installer script

`latexindent.pl` ships with a helper script that will install any missing `perl` modules on your system; if you run

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

or

```
C:\Users\cmh>perl latexindent-module-installer.pl
```

then, once you have answered `Y`, the appropriate modules will be installed onto your distribution.

A.2 Manually installed modules

Manually installing the modules given in Listing 342 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 [12]; 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-5.22.1
cmh:~$ perlbrew switch perl-5.22.1
cmh:~$ sudo apt-get install curl
cmh:~$ curl -L http://cpanmin.us | perl - App::cpanminus
cmh:~$ cpanm YAML::Tiny
cmh:~$ cpanm File::HomeDir
cmh:~$ cpanm Unicode::GCString
cmh:~$ cpanm Log::Log4perl
cmh:~$ cpanm Log::Dispatch
```

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` variables. Thank you to [7] for this feature. 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
```

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 logFilePreferences

Listing 16 on page 19 describes the options for customising the information given to the log file, and we provide a few demonstrations here. Let's say that we start with the code given in Listing 343, and the settings specified in Listing 344.

LISTING 343: simple.tex

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

LISTING 344: logfile-prefs1.yaml

```
logFilePreferences:
    showDecorationStartCodeBlockTrace: "+++++"
    showDecorationFinishCodeBlockTrace: "-----"
```

If we run the following command (noting that `-t` is active)

```
cmh:~$ latexindent.pl -t -l=logfile-prefs1.yaml simple.tex
```

then on inspection of `indent.log` we will find the snippet given in Listing 345.

LISTING 345: indent.log

```
       +++++
TRACE: environment found: myenv
       No ancestors found for myenv
       Storing settings for myenvenvironments
       indentRulesGlobal specified (0) for environments, ...
       Using defaultIndent for myenv
       Putting linebreak after replacementText for myenv
       looking for COMMANDS and key = {value}
TRACE: Searching for commands with optional and/or mandatory arguments AND key =
    {value}
       looking for SPECIAL begin/end
TRACE: Searching myenv for special begin/end (see specialBeginEnd)
TRACE: Searching myenv for optional and mandatory arguments
       ... no arguments found
       -----
```

Notice that the information given about `myenv` is 'framed' using `+++++` and `-----` respectively.

D 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 346 are *obsolete* from Version 3.0 onwards.

LISTING 346: 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 347 and 348

LISTING 347: `indentAfterThisHeading` in Version 2.2

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

LISTING 348: `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 349; as of Version 3.0, you would write YAML as in Listing 350 or, if you're using `-m` switch, Listing 351.

LISTING 349: `noAdditionalIndent` in Version 2.2

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

LISTING 350: `noAdditionalIndent` for `displayMath` in Version 3.0

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

LISTING 351: `noAdditionalIndent` for `displayMath` in Version 3.0

```
noAdditionalIndent:
    displayMath: 1
```

End