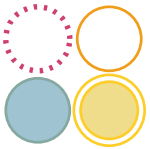
latexindent.pl

Version 3.10



Chris Hughes *

2021-06-19

`latexindent.pl` is a `Perl` script that indents `.tex` (and other) files according to an indentation scheme that the user can modify to suit their taste. Environments, including those with alignment delimiters (such as `tabular`), and commands, including those that can split braces and brackets across lines, are *usually* handled correctly by the script. Options for `verbatim`-like environments and commands, together with indentation after headings (such as `chapter`, `section`, etc) are also available. The script also has the ability to modify line breaks, and to add comment symbols and blank lines; furthermore, it permits string or regex-based substitutions. All user options are customisable via the switches and the YAML interface; you can find a quick start guide in Section 1.4 on page 11.

Contents

1 Introduction **10**
1.1 Thanks . . . 10
1.2 License . . . 10
1.3 About this documentation . . . 10
1.4 Quick start . . . 11
1.5 A word about regular expressions . . . 12

2 Demonstration: before and after **13**

3 How to use the script **14**
3.1 From the command line . . . 14
3.2 From arara . . . 20

4 indentconfig.yaml, local settings and the -y switch **21**
4.1 indentconfig.yaml and .indentconfig.yaml . . . 21
4.2 localSettings.yaml and friends . . . 22
4.3 The -y|yaml switch . . . 23
4.4 Settings load order . . . 23

5 defaultSettings.yaml **25**
5.1 Backup and log file preferences . . . 25

*and contributors! See Section 10.2 on page 130. For all communication, please visit [11].

5.2 Verbatim code blocks . . . 27
5.3 filecontents and preamble . . . 29
5.4 Indentation and horizontal space . . . 29
5.5 Aligning at delimiters . . . 30
5.5.1 lookForAlignDelims: spacesBeforeAmpersand . . . 34
5.5.2 lookForAlignDelims: alignFinalDoubleBackSlash . . . 36
5.5.3 lookForAlignDelims: the dontMeasure feature . . . 37
5.5.4 lookForAlignDelims: the delimiterRegEx and delimiterJustification feature . . . 39
5.6 Indent after items, specials and headings . . . 41
5.7 The code blocks known latexindent.pl . . . 48
5.8 noAdditionalIndent and indentRules . . . 48
5.8.1 Environments and their arguments . . . 50
5.8.2 Environments with items . . . 56
5.8.3 Commands with arguments . . . 57
5.8.4 ifelsefi code blocks . . . 59
5.8.5 specialBeginEnd code blocks . . . 61
5.8.6 afterHeading code blocks . . . 62
5.8.7 The remaining code blocks . . . 64
5.8.8 Summary . . . 66
5.9 Commands and the strings between their arguments . . . 66

6 The -m (modifylinebreaks) switch 72
6.1 textWrapOptions: modifying line breaks by text wrapping . . . 73
6.1.1 text wrapping on a per-code-block basis . . . 76
6.1.2 Summary of text wrapping . . . 82
6.2 oneSentencePerLine: modifying line breaks for sentences . . . 82
6.2.1 sentencesFollow . . . 84
6.2.2 sentencesBeginWith . . . 85
6.2.3 sentencesEndWith . . . 86
6.2.4 Features of the oneSentencePerLine routine . . . 88
6.2.5 text wrapping and indenting sentences . . . 89
6.3 removeParagraphLineBreaks: modifying line breaks for paragraphs . . . 91
6.4 Combining removeParagraphLineBreaks and textWrapOptions . . . 97
6.5 Poly-switches . . . 98
6.6 modifyLineBreaks for environments . . . 99
6.6.1 Adding line breaks: BeginStartsOnOwnLine and BodyStartsOnOwnLine . . . 99
6.6.2 Adding line breaks using EndStartsOnOwnLine and EndFinishesWithLineBreak 101
6.6.3 poly-switches 1, 2, and 3 only add line breaks when necessary . . . 102
6.6.4 Removing line breaks (poly-switches set to −1) . . . 103
6.6.5 About trailing horizontal space . . . 104
6.6.6 poly-switch line break removal and blank lines . . . 105
6.7 Poly-switches for double back slash . . . 106
6.7.1 Double back slash starts on own line . . . 106
6.7.2 Double back slash finishes with line break . . . 107
6.7.3 Double back slash poly-switches for specialBeginEnd . . . 107
6.7.4 Double back slash poly-switches for optional and mandatory arguments . . . 108
6.7.5 Double back slash optional square brackets . . . 109
6.8 Poly-switches for other code blocks . . . 109
6.9 Partnering BodyStartsOnOwnLine with argument-based poly-switches . . . 111
6.10 Conflicting poly-switches: sequential code blocks . . . 112
6.11 Conflicting poly-switches: nested code blocks . . . 113

7 The -r, -rv and -rr switches 115
7.1 Introduction to replacements . . . 115
7.2 The two types of replacements . . . 116
7.3 Examples of replacements . . . 116

8 Fine tuning 124

9 Conclusions and known limitations 129

10 References 130
10.1 External links . . . 130
10.2 Contributors . . . 130

A Required Perl modules 132
A.1 Module installer script . . . 132
A.2 Manually installed modules . . . 132
A.2.1 Linux . . . 133
A.2.2 Mac . . . 133
A.2.3 Windows . . . 134

B Updating the path variable 135
B.1 Add to path for Linux . . . 135
B.2 Add to path for Windows . . . 135

C logFilePreferences 137

D Encoding indentconfig.yaml 138

E dos2unix linebreak adjustment 139

F Differences from Version 2.2 to 3.0 140

Index 142

Listings

LISTING 1: demo-tex.tex · · · · · 10
LISTING 2: fileExtensionPreference · · · · · 11
LISTING 3: modifyLineBreaks · · · · · 11
LISTING 4: replacements · · · · · 11
LISTING 5: Possible error messages · · · · · 11
LISTING 6: filecontents1.tex · · · · · 13
LISTING 7: filecontents1.tex default output · · · · · 13
LISTING 8: tikzset.tex · · · · · 13
LISTING 9: tikzset.tex default output · · · · · 13
LISTING 10: pstricks.tex · · · · · 13
LISTING 11: pstricks.tex default output · · · · · 13
LISTING 14: The encoding option for indentconfig.yaml · · · · · 22
LISTING 16: fileExtensionPreference · · · · · 25
LISTING 17: logFilePreferences · · · · · 26
LISTING 18: verbatimEnvironments · · · · · 27
LISTING 19: verbatimCommands · · · · · 27
LISTING 20: noIndentBlock · · · · · 27
LISTING 21: noIndentBlock.tex · · · · · 27
LISTING 22: noIndentBlock1.tex · · · · · 28
LISTING 23: noindent1.yaml · · · · · 28
LISTING 24: noindent2.yaml · · · · · 28
LISTING 25: noindent3.yaml · · · · · 28
LISTING 26: noIndentBlock1.tex using Listing 23 or Listing 24 · · · · · 28
LISTING 27: noIndentBlock1.tex using Listing 25 · · · 28
LISTING 28: fileContentsEnvironments · · · · · 29
LISTING 29: lookForPreamble · · · · · 29
LISTING 30: Motivating preambleCommandsBeforeEnvironments 29
LISTING 31: removeTrailingWhitespace · · · · · 30
LISTING 34: tabular1.tex · · · · · 30
LISTING 35: tabular1.tex default output · · · · · 30
LISTING 36: lookForAlignDelims (advanced) · · · · · 31
LISTING 37: tabular2.tex · · · · · 32
LISTING 38: tabular2.yaml · · · · · 32

Listing 39: tabular3.yaml 32
Listing 40: tabular4.yaml 32
Listing 41: tabular5.yaml 32
Listing 42: tabular6.yaml 32
Listing 43: tabular7.yaml 32
Listing 44: tabular8.yaml 32
Listing 45: tabular2.tex default output 32
Listing 46: tabular2.tex using Listing 38 33
Listing 47: tabular2.tex using Listing 39 33
Listing 48: tabular2.tex using Listings 38 and 40 33
Listing 49: tabular2.tex using Listings 38 and 41 33
Listing 50: tabular2.tex using Listings 38 and 42 33
Listing 51: tabular2.tex using Listings 38 and 43 34
Listing 52: tabular2.tex using Listings 38 and 44 34
Listing 53: aligned1.tex 35
Listing 54: aligned1-default.tex 35
Listing 55: sba1.yaml 35
Listing 56: sba2.yaml 35
Listing 57: sba3.yaml 35
Listing 58: sba4.yaml 35
Listing 59: aligned1-mod1.tex 35
Listing 60: sba5.yaml 36
Listing 61: sba6.yaml 36
Listing 62: aligned1-mod5.tex 36
Listing 63: aligned1.tex using Listing 64 36
Listing 64: sba7.yaml 36
Listing 65: tabular4.tex 36
Listing 66: tabular4-default.tex 36
Listing 67: tabular4-FDBS.tex 36
Listing 68: matrix1.tex 37
Listing 69: matrix1.tex default output 37
Listing 70: align-block.tex 37
Listing 71: align-block.tex default output 37
Listing 72: tabular-DM.tex 37
Listing 73: tabular-DM.tex default output 37
Listing 74: tabular-DM.tex using Listing 75 37
Listing 75: dontMeasure1.yaml 37
Listing 76: tabular-DM.tex using Listing 77 or Listing 79 38
Listing 77: dontMeasure2.yaml 38
Listing 78: tabular-DM.tex using Listing 79 or Listing 79 38
Listing 79: dontMeasure3.yaml 38
Listing 80: dontMeasure4.yaml 38
Listing 81: tabular-DM.tex using Listing 82 39
Listing 82: dontMeasure5.yaml 39
Listing 83: tabular-DM.tex using Listing 84 39
Listing 84: dontMeasure6.yaml 39
Listing 85: tabbing.tex 39
Listing 86: tabbing.tex default output 39
Listing 87: tabbing.tex using Listing 88 40
Listing 88: delimiterRegEx1.yaml 40
Listing 89: tabbing.tex using Listing 90 40
Listing 90: delimiterRegEx2.yaml 40
Listing 91: tabbing.tex using Listing 92 40
Listing 92: delimiterRegEx3.yaml 40
Listing 93: tabbing1.tex 41
Listing 94: tabbing1-mod4.tex 41
Listing 95: delimiterRegEx4.yaml 41
Listing 96: tabbing1-mod5.tex 41
Listing 97: delimiterRegEx5.yaml 41
Listing 98: indentAfterItems 41
Listing 99: items1.tex 41
Listing 100: items1.tex default output 41
Listing 101: itemNames 41
Listing 102: specialBeginEnd 42
Listing 103: special1.tex before 42
Listing 104: special1.tex default output 42
Listing 105: specialLR.tex 42
Listing 106: specialsLeftRight.yaml 43
Listing 107: specialBeforeCommand.yaml 43
Listing 108: specialLR.tex using Listing 106 43
Listing 109: specialLR.tex using Listings 106 and 107 43
Listing 110: special2.tex 43
Listing 111: middle.yaml 44
Listing 112: special2.tex using Listing 111 44
Listing 113: middle1.yaml 44
Listing 114: special2.tex using Listing 113 44
Listing 115: special-verb1.yaml 44
Listing 116: special3.tex and output using Listing 115 44
Listing 117: special-align.tex 45
Listing 118: edge-node1.yaml 45
Listing 119: special-align.tex using Listing 118 45
Listing 120: edge-node2.yaml 45
Listing 121: special-align.tex using Listing 120 45
Listing 122: indentAfterHeadings 46
Listing 123: headings1.yaml 46
Listing 124: headings1.tex 46
Listing 125: headings1.tex using Listing 123 47
Listing 126: headings1.tex second modification 47
Listing 127: mult-nested.tex 47
Listing 128: mult-nested.tex default output 47
Listing 129: max-indentation1.yaml 48
Listing 130: mult-nested.tex using Listing 129 48
Listing 131: myenv.tex 50
Listing 132: myenv-noAdd1.yaml 50
Listing 133: myenv-noAdd2.yaml 50

LISTING 134: myenv.tex output (using either Listing 132 or Listing 133) 50
LISTING 135: myenv-noAdd3.yaml 51
LISTING 136: myenv-noAdd4.yaml 51
LISTING 137: myenv.tex output (using either Listing 135 or Listing 136) 51
LISTING 138: myenv-args.tex 51
LISTING 139: myenv-args.tex using Listing 132 51
LISTING 140: myenv-noAdd5.yaml 52
LISTING 141: myenv-noAdd6.yaml 52
LISTING 142: myenv-args.tex using Listing 140 52
LISTING 143: myenv-args.tex using Listing 141 52
LISTING 144: myenv-rules1.yaml 52
LISTING 145: myenv-rules2.yaml 52
LISTING 146: myenv.tex output (using either Listing 144 or Listing 145) 53
LISTING 147: myenv-args.tex using Listing 144 53
LISTING 148: myenv-rules3.yaml 53
LISTING 149: myenv-rules4.yaml 53
LISTING 150: myenv-args.tex using Listing 148 54
LISTING 151: myenv-args.tex using Listing 149 54
LISTING 152: noAdditionalIndentGlobal 54
LISTING 153: myenv-args.tex using Listing 152 54
LISTING 154: myenv-args.tex using Listings 144 and 152 54
LISTING 155: opt-args-no-add-glob.yaml 55
LISTING 156: mand-args-no-add-glob.yaml 55
LISTING 157: myenv-args.tex using Listing 155 55
LISTING 158: myenv-args.tex using Listing 156 55
LISTING 159: indentRulesGlobal 55
LISTING 160: myenv-args.tex using Listing 159 56
LISTING 161: myenv-args.tex using Listings 144 and 159 56
LISTING 162: opt-args-indent-rules-glob.yaml .. 56
LISTING 163: mand-args-indent-rules-glob.yaml 56
LISTING 164: myenv-args.tex using Listing 162 56
LISTING 165: myenv-args.tex using Listing 163 56
LISTING 166: item-noAdd1.yaml 56
LISTING 167: item-rules1.yaml 56
LISTING 168: items1.tex using Listing 166 57
LISTING 169: items1.tex using Listing 167 57
LISTING 170: items-noAdditionalGlobal.yaml 57
LISTING 171: items-indentRulesGlobal.yaml 57
LISTING 172: mycommand.tex 57
LISTING 173: mycommand.tex default output 57
LISTING 174: mycommand-noAdd1.yaml 58
LISTING 175: mycommand-noAdd2.yaml 58
LISTING 176: mycommand.tex using Listing 174 58
LISTING 177: mycommand.tex using Listing 175 58
LISTING 178: mycommand-noAdd3.yaml 58
LISTING 179: mycommand-noAdd4.yaml 58
LISTING 180: mycommand.tex using Listing 178 59
LISTING 181: mycommand.tex using Listing 179 59
LISTING 182: mycommand-noAdd5.yaml 59
LISTING 183: mycommand-noAdd6.yaml 59
LISTING 184: mycommand.tex using Listing 182 59
LISTING 185: mycommand.tex using Listing 183 59
LISTING 186: ifelsefi1.tex 60
LISTING 187: ifelsefi1.tex default output 60
LISTING 188: ifnum-noAdd.yaml 60
LISTING 189: ifnum-indent-rules.yaml 60
LISTING 190: ifelsefi1.tex using Listing 188 60
LISTING 191: ifelsefi1.tex using Listing 189 60
LISTING 192: ifelsefi-noAdd-glob.yaml 60
LISTING 193: ifelsefi-indent-rules-global.yaml 60
LISTING 194: ifelsefi1.tex using Listing 192 61
LISTING 195: ifelsefi1.tex using Listing 193 61
LISTING 196: ifelsefi2.tex 61
LISTING 197: ifelsefi2.tex default output 61
LISTING 198: displayMath-noAdd.yaml 61
LISTING 199: displayMath-indent-rules.yaml 61
LISTING 200: special1.tex using Listing 198 62
LISTING 201: special1.tex using Listing 199 62
LISTING 202: special-noAdd-glob.yaml 62
LISTING 203: special-indent-rules-global.yaml 62
LISTING 204: special1.tex using Listing 202 62
LISTING 205: special1.tex using Listing 203 62
LISTING 206: headings2.tex 62
LISTING 207: headings2.tex using Listing 208 63
LISTING 208: headings3.yaml 63
LISTING 209: headings2.tex using Listing 210 63
LISTING 210: headings4.yaml 63
LISTING 211: headings2.tex using Listing 212 63
LISTING 212: headings5.yaml 63
LISTING 213: headings2.tex using Listing 214 63
LISTING 214: headings6.yaml 63
LISTING 215: headings2.tex using Listing 216 64
LISTING 216: headings7.yaml 64
LISTING 217: headings2.tex using Listing 218 64
LISTING 218: headings8.yaml 64
LISTING 219: headings2.tex using Listing 220 64
LISTING 220: headings9.yaml 64
LISTING 221: pgfkeys1.tex 64
LISTING 222: pgfkeys1.tex default output 64
LISTING 223: child1.tex 65
LISTING 224: child1.tex default output 65
LISTING 225: psforeach1.tex 65
LISTING 226: psforeach1.tex default output 65

LISTING 227: noAdditionalIndentGlobal 66
LISTING 228: indentRulesGlobal 66
LISTING 229: commandCodeBlocks 67
LISTING 230: pstricks1.tex 67
LISTING 231: pstricks1 default output 67
LISTING 232: pstricks1.tex using Listing 233 67
LISTING 233: noRoundParentheses.yaml 67
LISTING 234: pstricks1.tex using Listing 235 68
LISTING 235: defFunction.yaml 68
LISTING 236: tikz-node1.tex 68
LISTING 237: tikz-node1 default output 68
LISTING 238: tikz-node1.tex using Listing 239 69
LISTING 239: draw.yaml 69
LISTING 240: tikz-node1.tex using Listing 241 69
LISTING 241: no-strings.yaml 69
LISTING 242: amalgamate-demo.yaml 69
LISTING 243: amalgamate-demo1.yaml 69
LISTING 244: amalgamate-demo2.yaml 69
LISTING 245: amalgamate-demo3.yaml 70
LISTING 246: for-each.tex 70
LISTING 247: for-each default output 70
LISTING 248: for-each.tex using Listing 249 70
LISTING 249: foreach.yaml 70
LISTING 250: ifnextchar.tex 70
LISTING 251: ifnextchar.tex default output 70
LISTING 252: ifnextchar.tex using Listing 253 71
LISTING 253: no-ifnextchar.yaml 71
LISTING 254: modifyLineBreaks 73
LISTING 255: mlb1.tex 73
LISTING 256: mlb1.tex out output 73
LISTING 257: textWrapOptions 73
LISTING 258: textwrap1.tex 74
LISTING 259: textwrap1-mod1.tex 74
LISTING 260: textwrap1.yaml 74
LISTING 261: textwrap2.tex 74
LISTING 262: textwrap2-mod1.tex 75
LISTING 263: textwrap3.tex 75
LISTING 264: textwrap3-mod1.tex 75
LISTING 265: textwrap4.tex 75
LISTING 266: textwrap4-mod2.tex 76
LISTING 267: textwrap2.yaml 76
LISTING 268: textwrap4-mod2A.tex 76
LISTING 269: textwrap2A.yaml 76
LISTING 270: textwrap4-mod2B.tex 76
LISTING 271: textwrap2B.yaml 76
LISTING 272: textwrap-ts.tex 76
LISTING 273: tabstop.yaml 76
LISTING 274: textwrap-ts-mod1.tex 76
LISTING 275: textWrapOptions 77
LISTING 276: textwrap5.tex 77
LISTING 277: textwrap3.yaml 77
LISTING 278: textwrap4.yaml 77
LISTING 279: textwrap5.yaml 77
LISTING 280: textwrap5-mod3.tex 78
LISTING 281: textwrap6.tex 78
LISTING 282: textwrap6.tex using Listing 279 79
LISTING 283: textwrap6.yaml 79
LISTING 284: textwrap7.yaml 79
LISTING 285: textwrap8.yaml 79
LISTING 286: textwrap6.tex using Listing 283 80
LISTING 287: textwrap6.tex using Listing 284 80
LISTING 288: textwrap6.tex using Listing 285 80
LISTING 289: textwrap9.yaml 81
LISTING 290: textwrap10.yaml 81
LISTING 291: textwrap11.yaml 81
LISTING 292: textwrap6.tex using Listing 289 81
LISTING 293: textwrap6.tex using Listing 291 82
LISTING 294: oneSentencePerLine 83
LISTING 295: multiple-sentences.tex 83
LISTING 296: multiple-sentences.tex using Listing 297 84
LISTING 297: manipulate-sentences.yaml 84
LISTING 298: multiple-sentences.tex using Listing 299 84
LISTING 299: keep-sen-line-breaks.yaml 84
LISTING 300: sentencesFollow 84
LISTING 301: sentencesBeginWith 84
LISTING 302: sentencesEndWith 84
LISTING 303: multiple-sentences.tex using Listing 304 85
LISTING 304: sentences-follow1.yaml 85
LISTING 305: multiple-sentences1.tex 85
LISTING 306: multiple-sentences1.tex using Listing 297 on page 84 85
LISTING 307: multiple-sentences1.tex using Listing 308 85
LISTING 308: sentences-follow2.yaml 85
LISTING 309: multiple-sentences2.tex 86
LISTING 310: multiple-sentences2.tex using Listing 297 on page 84 86
LISTING 311: multiple-sentences2.tex using Listing 312 86
LISTING 312: sentences-begin1.yaml 86
LISTING 313: multiple-sentences.tex using Listing 314 86
LISTING 314: sentences-end1.yaml 86
LISTING 315: multiple-sentences.tex using Listing 316 87
LISTING 316: sentences-end2.yaml 87
LISTING 317: url.tex 87

LISTING 318: `url.tex` using Listing 297 on page 84 ··· 87
LISTING 319: `url.tex` using Listing 320 ··· 88
LISTING 320: `alt-full-stop1.yaml` ··· 88
LISTING 321: `multiple-sentences3.tex` ··· 88
LISTING 322: `multiple-sentences3.tex` using Listing 297 on page 84 ··· 88
LISTING 323: `multiple-sentences4.tex` ··· 89
LISTING 324: `multiple-sentences4.tex` using Listing 297 on page 84 ··· 89
LISTING 325: `multiple-sentences4.tex` using Listing 299 on page 84 ··· 89
LISTING 326: `multiple-sentences4.tex` using Listing 327 ··· 89
LISTING 327: `item-rules2.yaml` ··· 89
LISTING 328: `multiple-sentences5.tex` ··· 89
LISTING 329: `multiple-sentences5.tex` using Listing 330 ··· 90
LISTING 330: `sentence-wrap1.yaml` ··· 90
LISTING 331: `multiple-sentences6.tex` ··· 90
LISTING 332: `multiple-sentences6-mod1.tex` using Listing 330 ··· 90
LISTING 333: `multiple-sentences6-mod2.tex` using Listing 330 and no sentence indentation ··· 90
LISTING 334: `itemize.yaml` ··· 91
LISTING 335: `multiple-sentences6-mod3.tex` using Listing 330 and Listing 334 ··· 91
LISTING 336: `removeParagraphLineBreaks` ··· 91
LISTING 337: `shortlines.tex` ··· 92
LISTING 338: `remove-para1.yaml` ··· 92
LISTING 339: `shortlines1.tex` ··· 92
LISTING 340: `shortlines1-tws.tex` ··· 92
LISTING 341: `shortlines-mand.tex` ··· 93
LISTING 342: `shortlines-opt.tex` ··· 93
LISTING 343: `shortlines-mand1.tex` ··· 93
LISTING 344: `shortlines-opt1.tex` ··· 93
LISTING 345: `shortlines-envs.tex` ··· 94
LISTING 346: `remove-para2.yaml` ··· 94
LISTING 347: `remove-para3.yaml` ··· 94
LISTING 348: `shortlines-envs2.tex` ··· 94
LISTING 349: `shortlines-envs3.tex` ··· 95
LISTING 350: `shortlines-md.tex` ··· 95
LISTING 351: `remove-para4.yaml` ··· 95
LISTING 352: `shortlines-md4.tex` ··· 96
LISTING 353: `paragraphsStopAt` ··· 96
LISTING 354: `sl-stop.tex` ··· 96
LISTING 355: `stop-command.yaml` ··· 96
LISTING 356: `stop-comment.yaml` ··· 96
LISTING 357: `sl-stop4.tex` ··· 97
LISTING 358: `sl-stop4-command.tex` ··· 97
LISTING 359: `sl-stop4-comment.tex` ··· 97
LISTING 360: `textwrap7.tex` ··· 97
LISTING 361: `textwrap7.tex` using Listing 277 ··· 98
LISTING 362: `textwrap7-mod12.tex` ··· 98
LISTING 363: `textwrap12.yaml` ··· 98
LISTING 364: `environments` ··· 99
LISTING 365: `env-mlb1.tex` ··· 99
LISTING 366: `env-mlb1.yaml` ··· 99
LISTING 367: `env-mlb2.yaml` ··· 99
LISTING 368: `env-mlb.tex` using Listing 366 ··· 99
LISTING 369: `env-mlb.tex` using Listing 367 ··· 99
LISTING 370: `env-mlb3.yaml` ··· 99
LISTING 371: `env-mlb4.yaml` ··· 99
LISTING 372: `env-mlb.tex` using Listing 370 ··· 100
LISTING 373: `env-mlb.tex` using Listing 371 ··· 100
LISTING 374: `env-mlb5.yaml` ··· 100
LISTING 375: `env-mlb6.yaml` ··· 100
LISTING 376: `env-mlb.tex` using Listing 374 ··· 100
LISTING 377: `env-mlb.tex` using Listing 375 ··· 100
LISTING 378: `env-beg4.yaml` ··· 100
LISTING 379: `env-body4.yaml` ··· 100
LISTING 380: `env-mlb1.tex` ··· 100
LISTING 381: `env-mlb1.tex` using Listing 378 ··· 100
LISTING 382: `env-mlb1.tex` using Listing 379 ··· 100
LISTING 383: `env-mlb7.yaml` ··· 101
LISTING 384: `env-mlb8.yaml` ··· 101
LISTING 385: `env-mlb.tex` using Listing 383 ··· 101
LISTING 386: `env-mlb.tex` using Listing 384 ··· 101
LISTING 387: `env-mlb9.yaml` ··· 101
LISTING 388: `env-mlb10.yaml` ··· 101
LISTING 389: `env-mlb.tex` using Listing 387 ··· 101
LISTING 390: `env-mlb.tex` using Listing 388 ··· 101
LISTING 391: `env-mlb11.yaml` ··· 101
LISTING 392: `env-mlb12.yaml` ··· 101
LISTING 393: `env-mlb.tex` using Listing 391 ··· 102
LISTING 394: `env-mlb.tex` using Listing 392 ··· 102
LISTING 395: `env-end4.yaml` ··· 102
LISTING 396: `env-end-f4.yaml` ··· 102
LISTING 397: `env-mlb1.tex` using Listing 395 ··· 102
LISTING 398: `env-mlb1.tex` using Listing 396 ··· 102
LISTING 399: `env-mlb2.tex` ··· 102
LISTING 400: `env-mlb3.tex` ··· 102
LISTING 401: `env-mlb3.tex` using Listing 367 on page 99 ··· 103
LISTING 402: `env-mlb3.tex` using Listing 371 on page 99 ··· 103
LISTING 403: `env-mlb4.tex` ··· 103
LISTING 404: `env-mlb13.yaml` ··· 103
LISTING 405: `env-mlb14.yaml` ··· 103
LISTING 406: `env-mlb15.yaml` ··· 103
LISTING 407: `env-mlb16.yaml` ··· 103
LISTING 408: `env-mlb4.tex` using Listing 404 ··· 104

LISTING 409: env-mlb4.tex using Listing 405 104
LISTING 410: env-mlb4.tex using Listing 406 104
LISTING 411: env-mlb4.tex using Listing 407 104
LISTING 412: env-mlb5.tex 104
LISTING 413: removeTWS-before.yaml 104
LISTING 414: env-mlb5.tex using Listings 408 to 411 . 104
LISTING 415: env-mlb5.tex using Listings 408 to 411 *and* Listing 413 105
LISTING 416: env-mlb6.tex 105
LISTING 417: UnpreserveBlankLines.yaml 105
LISTING 418: env-mlb6.tex using Listings 408 to 411 . 105
LISTING 419: env-mlb6.tex using Listings 408 to 411 *and* Listing 417 105
LISTING 420: env-mlb7.tex 105
LISTING 421: env-mlb7-preserve.tex 106
LISTING 422: env-mlb7-no-preserve.tex 106
LISTING 423: tabular3.tex 106
LISTING 424: tabular3.tex using Listing 425 106
LISTING 425: DBS1.yaml 106
LISTING 426: tabular3.tex using Listing 427 107
LISTING 427: DBS2.yaml 107
LISTING 428: tabular3.tex using Listing 429 107
LISTING 429: DBS3.yaml 107
LISTING 430: tabular3.tex using Listing 431 107
LISTING 431: DBS4.yaml 107
LISTING 432: special4.tex 108
LISTING 433: special4.tex using Listing 434 108
LISTING 434: DBS5.yaml 108
LISTING 435: mycommand2.tex 108
LISTING 436: mycommand2.tex using Listing 437 109
LISTING 437: DBS6.yaml 109
LISTING 438: mycommand2.tex using Listing 439 109
LISTING 439: DBS7.yaml 109
LISTING 440: pmatrix3.tex 109
LISTING 441: pmatrix3.tex using Listing 429 109
LISTING 442: mycommand1.tex 111
LISTING 443: mycommand1.tex using Listing 444 111
LISTING 444: mycom-mlb1.yaml 111
LISTING 445: mycommand1.tex using Listing 446 111
LISTING 446: mycom-mlb2.yaml 111
LISTING 447: mycommand1.tex using Listing 448 112
LISTING 448: mycom-mlb3.yaml 112
LISTING 449: mycommand1.tex using Listing 450 112
LISTING 450: mycom-mlb4.yaml 112
LISTING 451: mycommand1.tex using Listing 452 112
LISTING 452: mycom-mlb5.yaml 112
LISTING 453: mycommand1.tex using Listing 454 113
LISTING 454: mycom-mlb6.yaml 113
LISTING 455: nested-env.tex 113
LISTING 456: nested-env.tex using Listing 457 113
LISTING 457: nested-env-mlb1.yaml 113
LISTING 458: nested-env.tex using Listing 459 114
LISTING 459: nested-env-mlb2.yaml 114
LISTING 460: replacements 115
LISTING 461: replace1.tex 116
LISTING 462: replace1.tex default 116
LISTING 463: replace1.tex using Listing 464 116
LISTING 464: replace1.yaml 116
LISTING 465: colsep.tex 116
LISTING 466: colsep.tex using Listing 465 117
LISTING 467: colsep.yaml 117
LISTING 468: colsep.tex using Listing 469 117
LISTING 469: colsep1.yaml 117
LISTING 470: colsep.tex using Listing 471 118
LISTING 471: multi-line.yaml 118
LISTING 472: colsep.tex using Listing 473 118
LISTING 473: multi-line1.yaml 118
LISTING 474: displaymath.tex 119
LISTING 475: displaymath.tex using Listing 476 119
LISTING 476: displaymath1.yaml 119
LISTING 477: displaymath.tex using Listings 476 and 478 120
LISTING 478: equation.yaml 120
LISTING 479: phrase.tex 120
LISTING 480: phrase.tex using Listing 481 120
LISTING 481: hspace.yaml 120
LISTING 482: references.tex 121
LISTING 483: references.tex using Listing 484 121
LISTING 484: reference.yaml 121
LISTING 485: verb1.tex 121
LISTING 486: verbatim1.yaml 121
LISTING 487: verb1-mod1.tex 122
LISTING 488: verb1-rv-mod1.tex 122
LISTING 489: verb1-rr-mod1.tex 122
LISTING 490: amalg1.tex 122
LISTING 491: amalg1-yaml.yaml 122
LISTING 492: amalg2-yaml.yaml 122
LISTING 493: amalg3-yaml.yaml 122
LISTING 494: amalg1.tex using Listing 491 123
LISTING 495: amalg1.tex using Listings 491 and 492 . 123
LISTING 496: amalg1.tex using Listings 491 to 493 ... 123
LISTING 497: fineTuning 124
LISTING 498: finetuning1.tex 125
LISTING 499: finetuning1.tex default 125
LISTING 500: finetuning1.tex using Listing 501 126
LISTING 501: finetuning1.yaml 126
LISTING 502: finetuning2.tex 126
LISTING 503: finetuning2.tex default 126
LISTING 504: finetuning2.tex using Listing 505 126
LISTING 505: finetuning2.yaml 126

LISTING 506: `finetuning3.tex` 127
LISTING 507: `finetuning3.tex` using -y switch 127
LISTING 508: `finetuning4.tex` 127
LISTING 509: `href1.yaml` 127
LISTING 510: `href2.yaml` 127
LISTING 511: `finetuning4.tex` using Listing 509 127
LISTING 512: `finetuning4.tex` using Listing 510 128
LISTING 513: `href3.yaml` 128
LISTING 514: `matrix2.tex` 129
LISTING 515: `matrix2.tex` default output 129
LISTING 516: `helloworld.pl` 132
LISTING 517: `alpine-install.sh` 133
LISTING 518: `simple.tex` 137
LISTING 519: `logfile-prefs1.yaml` 137
LISTING 520: `indent.log` 137
LISTING 521: `encoding` demonstration for `indentconfig.yaml` 138
LISTING 522: Obsolete YAML fields from Version 3.0 140
LISTING 523: `indentAfterThisHeading` in Version 2.2 140
LISTING 524: `indentAfterThisHeading` in Version 3.0 140
LISTING 525: `noAdditionalIndent` in Version 2.2 141
LISTING 526: `noAdditionalIndent` for `displayMath` in Version 3.0 141
LISTING 527: `noAdditionalIndent` for `displayMath` in Version 3.0 141

SECTION 1

Introduction

1.1 Thanks

I first created `latexindent.pl` to help me format chapter files in a big project. After I blogged about it on the TEX stack exchange [1] I received some positive feedback and follow-up feature requests. A big thank you to Harish Kumar [15] 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 10.2 on page 130 for their valued contributions, and thank you to those who report bugs and request features at [11].

1.2 License

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

Before you start using it on any important files, bear in mind that `latexindent.pl` has the option to overwrite your `.tex` files. It will always make at least one backup (you can choose how many it makes, see page 25) 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 9). 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 [11] 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 [11].

If you have used any version 2. of `latexindent.pl`, there are a few changes to the interface; see appendix F on page 140 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 527. 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

```
41 fileExtensionPreference:
42     .tex: 1
43     .sty: 2
44     .cls: 3
45     .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 25.

LISTING 3: modifyLineBreaks `-m`

```
483 modifyLineBreaks:
484     preserveBlankLines: 1
485     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 72 for more details.

LISTING 4: replacements `-r`

```
614 replacements:
615   -
616     amalgamate: 1
617   -
618     this: 'latexindent.pl'
619     that: 'pl.latexindent'
620     lookForThis: 1
621     when: before
```

This type of listing is a `.yaml` file, but it will only be relevant when the `-r` switch is active; see Section 7 on page 115 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 (V3.10) are on the following pages:

encoding option for indentconfig.yaml (N) 22
logFilePreferences updated to include Dumper options (U) 27
noIndentBlock specified as regex (N) 27
spacesBeforeAmpersand leading blank column upgrade (U) 34
lookForThis optional for specialBeginEnd (U) 45
dos2unix linebreaks (N) 139

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 5, then you need to install the missing perl modules.

LISTING 5: 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 132.

1.5 A word about regular expressions

As you read this documentation, you may encounter the term *regular expressions*. I've tried to write this documentation in such a way so as to allow you to engage with them or not, as you prefer. This documentation is not designed to be a guide to regular expressions, and if you'd like to read about them, I recommend [10].

SECTION 2

Demonstration: before and after

Let's give a demonstration of some before and after code – after all, you probably won't want to try the script if you don't much like the results. You might also like to watch the video demonstration I made on youtube [27]

As you look at Listings 6 to 11, remember that `latexindent.pl` is just following its rules, and there is nothing particular about these code snippets. All of the rules can be modified so that you can personalise your indentation scheme.

In each of the samples given in Listings 6 to 11 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 6: `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 7: `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 8: `tikzset.tex`

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

LISTING 9: `tikzset.tex` default output

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

LISTING 10: `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 11: `pstricks.tex` default output

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

SECTION 3

How to use the script

`latexindent.pl` ships as part of the TEXLive distribution for Linux and Mac users; `latexindent.exe` ships as part of the TEXLive and MiKTEX distributions for Windows users. These files are also available from github [11] should you wish to use them without a TEX distribution; in this case, you may like to read appendix B on page 135 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 25.

`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 132 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 customised, but we will refer to the log file as `indent.log` throughout this document. There is a base of information that is written to `indent.log`, but other additional information will be written depending on which of the following options are used.

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

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

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

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 F on page 140 for details of how the interface has changed from Version 2.2 to Version 3.0 for this flag.

`-s, -silent`

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

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

`-t, -trace`

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

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

`-tt, -ttrace`

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

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

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

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

`latexindent.pl` will always load `defaultSettings.yaml` (rhymes with camel) and if it is called with the `-l` switch and it finds `localSettings.yaml` in the same directory as `myfile.tex`, then, if not found, it looks for `localSettings.yaml` (and friends, see Section 4.2 on page 22) in the current

U: 2021-03-14

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

The `-l` flag can take an *optional* parameter which details the name (or names separated by commas) of a YAML file(s) that resides in the same directory as `myfile.tex`; you can use this option if you would like to load a settings file in the current working directory that is *not* called `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 23.

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 23.

`-d, -onlydefault`

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

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

U: 2017-08-21

`-c, -cruft=<directory>`

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

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

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

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

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

N: 2021-05-07

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

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

`-sl, -screenlog`

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

N: 2018-01-13

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

`-m, -modifylinebreaks`

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

One of the most exciting developments in Version 3.0 is the ability to modify line breaks; for full details see Section 6 on page 72

`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 16 on page 25 for full details.

STDIN

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

N: 2018-01-13

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

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

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

U: 2018-01-13

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

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

Once you have finished typing your input, you can press

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

to signify that your input has finished. Thanks to [4] for an update to this feature.

-r, -replacement

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

N: 2019-07-13

You can call `latexindent.pl` with the `-r` switch to instruct it to perform replacements/substitutions on your file; full details and examples are given in Section 7 on page 115.

-rv, -replacementrespectverb

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

N: 2019-07-13

You can instruct `latexindent.pl` to perform replacements/substitutions by using the `-rv` switch, but will *respect verbatim code blocks*; full details and examples are given in Section 7 on page 115.

-rr, -onlyreplacement

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

N: 2019-07-13

You can instruct `latexindent.pl` to skip all of its other indentation operations and *only* perform replacements/substitutions by using the `-rr` switch; full details and examples are given in Section 7 on page 115.

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 for `latexindent.pl` and its associated documentation at [3].

SECTION 4

indentconfig.yaml, local settings and the -y switch

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

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 [9] 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 recognise 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`.

N: 2021-06-19

If you find that `latexindent.pl` does not read your YAML file, then it might be as a result of the default commandline encoding not being UTF-8; normally this will only occcur for Windows users. In this case, you might like to explore the `encoding` option for `indentconfig.yaml` as demonstrated in Listing 14.

LISTING 14: The `encoding` option for `indentconfig.yaml`

```
encoding: GB2312
paths:
- D:\cmh\latexindent.yaml
```

Thank you to [22] for this contribution; please see appendix D on page 138 and details within [21] for further information.

4.2 localSettings.yaml and friends

U: 2021-03-14

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

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

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

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

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

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

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

```
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 15, and you'll find plenty of further examples throughout this manual.

LISTING 15: `localSettings.yaml` (example)

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

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

4.3 The -y|yaml switch

N: 2017-08-21

You may use the `-y` switch to load your settings; for example, if you wished to specify the settings from Listing 15 using the `-y` switch, then you could use the following command:

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

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

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

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

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

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

If you wish to specify any regex-based settings using the `-y` switch, it is important not to use quotes surrounding the regex; for example, with reference to the 'one sentence per line' feature (Section 6.2 on page 82) and the listings within Listing 302 on page 84, the following settings give the option to have sentences end with a semicolon

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

4.4 Settings load order

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

1. `defaultSettings.yaml` is always loaded, and can not be renamed;

2. `anyUserSettings.yaml` and any other arbitrarily-named files specified in `indentconfig.yaml`;
3. `localSettings.yaml` but only if found in the same directory as `myfile.tex` and called with `-l` switch; this file can be renamed, provided that the call to `latexindent.pl` is adjusted accordingly (see Section 4.2). You may specify both relative and absolute paths to other YAML files using the `-l` switch, separating multiple files using commas;

U: 2017-08-21

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

N: 2017-08-21

A visual representation of this is given in Figure 1.

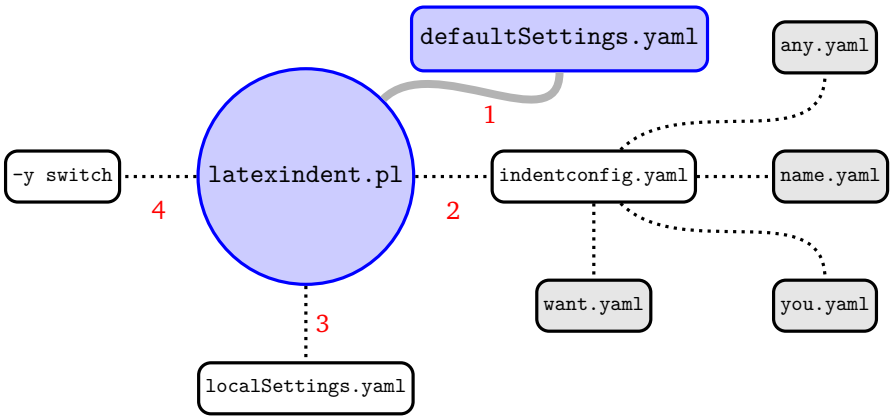


FIGURE 1: Schematic of the load order described in Section 4.4; solid lines represent mandatory files, dotted lines represent optional files. `indentconfig.yaml` can contain as many files as you like. The files will be loaded in order; if you specify settings for the same field in more than one file,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

SECTION 5

defaultSettings.yaml

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

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

`fileExtensionPreference`: ⟨*fields*⟩

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

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

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

LISTING 16: `fileExtensionPreference`

```
41 fileExtensionPreference:
42     .tex: 1
43     .sty: 2
44     .cls: 3
45     .bib: 4
```

Calling `latexindent.pl myfile` with the (default) settings specified in Listing 16 means that the script will first look for `myfile.tex`, then `myfile.sty`, `myfile.cls`, and finally `myfile.bib` in order[4].

5.1 Backup and log file preferences

`backupExtension`: ⟨*extension name*⟩

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

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

`onlyOneBackUp`: ⟨*integer*⟩

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

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

`maxNumberOfBackUps`: ⟨*integer*⟩

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

`cycleThroughBackUps`: ⟨*integer*⟩

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

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

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

`logFilePreferences`: ⟨*fields*⟩

`latexindent.pl` writes information to `indent.log`, some of which can be customized by changing `logFilePreferences`; see Listing 17. If you load your own user settings (see Section 4 on page 21) 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 17: `logFilePreferences`

```
85  logFilePreferences:
86      showEveryYamlRead: 1
87      showAmalgamatedSettings: 0
88      showDecorationStartCodeBlockTrace: 0
89      showDecorationFinishCodeBlockTrace: 0
90      endLogFileWith: '--------------'
91      showGitHubInfoFooter: 1
92      Dumper:
93        Terse: 1
94        Indent: 1
95        Useqq: 1
96        Deparse: 1
97        Quotekeys: 0
98        Sortkeys: 1
99        Pair: " => "
```

N: 2018-01-13

When either of the `trace` modes (see page 16) 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 137.

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

U: 2021-03-14

U: 2021-06-19

Note: `latexindent.pl` no longer uses the `log4perl` module to handle the creation of the logfile.

Some of the optionsfor Perl's `Dumper` module can be specified in Listing 17; see [7] and [6] for more information. These options will mostly be helpful for those calling `latexindent.pl` with the `-tt` option described in Section 3.1.

5.2 Verbatim code blocks

`verbatimEnvironments`: ⟨*fields*⟩

A field that contains a list of environments that you would like left completely alone – no indentation will be performed on environments that you have specified in this field, see Listing 18.

LISTING 18: `verbatimEnvironments`

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

LISTING 19: `verbatimCommands`

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

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

`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 72).

`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 20.

LISTING 20: `noIndentBlock`

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

Of course, you don't want to have to specify these as null environments in your code, so you use them with a comment symbol, %, followed by as many spaces (possibly none) as you like; see Listing 21 for example.

LISTING 21: `noIndentBlock.tex`

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

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

N: 2021-06-19

The `noIndentBlock` fields can also be specified in terms of `begin` and `end` fields. We begin with the file given in Listing 22

LISTING 22: noIndentBlock1.tex

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

The settings given in Listings 23 and 24 are equivalent

LISTING 23: noindent1.yaml

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

LISTING 24: noindent2.yaml

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

LISTING 25: noindent3.yaml

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

Upon running the commands

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

then we receive the output given in Listing 26.

LISTING 26: noIndentBlock1.tex using Listing 23 or Listing 24

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

The begin, body and end fields for noIndentBlock are all *regular expressions*. If the body field is not specified, then it takes a default value of .*? which is written explicitly in Listing 23.

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

Using Listing 25 demonstrates setting lookForThis to 0 (off); running the command

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

gives the output in Listing 27.

LISTING 27: noIndentBlock1.tex using Listing 25

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

We will demonstrate this feature later in the documentation in Listing 513.

5.3 filecontents and preamble

`fileContentsEnvironments`: ⟨*field*⟩

Before `latexindent.pl` determines the difference between preamble (if any) and the main document, it first searches for any of the environments specified in `fileContentsEnvironments`, see Listing 28. The behaviour of `latexindent.pl` on these environments is determined by their location (preamble or not), and the value `indentPreamble`, discussed next.

LISTING 28: `fileContentsEnvironments`

```
122 fileContentsEnvironments:
123     filecontents: 1
124     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 29, 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 29: lookForPreamble

```
130 lookForPreamble:
131     .tex: 1
132     .sty: 0
133     .cls: 0
134     .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 30.

LISTING 30: Motivating `preambleCommandsBeforeEnvironments`

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

5.4 Indentation and horizontal space

`defaultIndent`: ⟨*horizontal space*⟩

This is the default indentation used in the absence of other details for the code block with which we are working. The default value is `\t` which means a tab; we will explore customisation beyond `defaultIndent` in Section 5.8 on page 48.

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

`removeTrailingWhitespace`: ⟨*fields*⟩

Trailing white space can be removed both *before* and *after* processing the document, as detailed in Listing 31; each of the fields can take the values `0` or `1`. See Listings 413 to 415 on pages 104–105 for before and after results. Thanks to [28] for providing this feature.

LISTING 31: removeTrailingWhitespace

```
147 removeTrailingWhitespace:
148     beforeProcessing: 0
149     afterProcessing: 1
```

LISTING 32: removeTrailingWhitespace (alt)

```
removeTrailingWhitespace: 1
```

N: 2017-06-28

You can specify `removeTrailingWhitespace` simply as `0` or `1`, if you wish; in this case, `latexindent.pl` will set both `beforeProcessing` and `afterProcessing` to the value you specify; see Listing 32.

5.5 Aligning at delimiters

`lookForAlignDelims`: ⟨*fields*⟩

This contains a list of code blocks that are operated upon in a special way by `latexindent.pl` (see Listing 33). In fact, the fields in `lookForAlignDelims` can actually take two different forms: the *basic* version is shown in Listing 33 and the *advanced* version in Listing 36; we will discuss each in turn.

LISTING 33: lookForAlignDelims (basic)

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

Specifying code blocks in this field instructs `latexindent.pl` to try and align each column by its alignment delimiters. It does have some limitations (discussed further in Section 9), but in many cases it will produce results such as those in Listings 34 and 35.

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 20 on page 27).

LISTING 34: tabular1.tex

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

LISTING 35: 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 36 is for you.

LISTING 36: `lookForAlignDelims` (advanced)

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

Note that you can use a mixture of the basic and advanced form: in Listing 36 `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 33. If `delims` is set to 0 then the align at ampersand routine will not be called for this code block (default: 1);
- `alignDoubleBackSlash`: binary switch (0 or 1) to determine if \\ should be aligned (default: 1);
- U: 2018-01-13 `spacesBeforeDoubleBackSlash`: optionally, specifies the number (integer ≥ 0) of spaces to be inserted before \\ (default: 1);
- N: 2017-06-19 `multiColumnGrouping`: binary switch (0 or 1) that details if `latexindent.pl` should group columns above and below a `\multicolumn` command (default: 0);
- N: 2017-06-19 `alignRowsWithoutMaxDelims`: binary switch (0 or 1) that details if rows that do not contain the maximum number of 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);
- N: 2020-03-21 alignFinalDoubleBackSlash optionally specifies if the *final* double back slash should be used for alignment (default: 0);
- N: 2020-03-21 dontMeasure optionally specifies if user-specified cells, rows or the largest entries should *not* be measured (default: 0);
- N: 2020-03-21 delimiterRegEx optionally specifies the pattern matching to be used for the alignment delimeter (default: `'(?<!\\)(&)'`);
- N: 2020-03-21 delimiterJustification optionally specifies the justification for the alignment delimeters (default: left); note that this feature is only useful if you have delimiters of different lengths in the same column, discussed in Section 5.5.4.

We will explore most of these features using the file `tabular2.tex` in Listing 37 (which contains a `\multicolumn` command), and the YAML files in Listings 38 to 44; we will explore `alignFinalDoubleBackSlash`

in Listing 65; the dontMeasure feature will be described in Section 5.5.3, and delimiterRegEx in Section 5.5.4.

LISTING 37: 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 38: tabular2.yaml

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

LISTING 39: tabular3.yaml

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

LISTING 40: tabular4.yaml

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

LISTING 41: tabular5.yaml

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

LISTING 42: tabular6.yaml

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

LISTING 43: tabular7.yaml

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

LISTING 44: 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 45 to 52.

LISTING 45: 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 46: tabular2.tex using Listing 38

```
\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 47: tabular2.tex using Listing 39

```
\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 48: tabular2.tex using Listings 38 and 40

```
\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 49: tabular2.tex using Listings 38 and 41

```
\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 50: tabular2.tex using Listings 38 and 42

```
\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 51: tabular2.tex using Listings 38 and 43

```
\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 52: tabular2.tex using Listings 38 and 44

```
\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 45 and 46 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 45 the columns have been aligned at the ampersand;
- in Listing 46 the `\multicolumn` command has grouped the 2 columns beneath *and* above it, because `multiColumnGrouping` is set to 1 in Listing 38;
- in Listing 47 rows 3 and 6 have *not* been aligned at the ampersand, because `alignRowsWithoutMaxDelims` has been to set to 0 in Listing 39; however, the `\\` *have* still been aligned;
- in Listing 48 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 49 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 50 the `\\` have *not* been aligned, because `alignDoubleBackSlash` is set to `0`, otherwise the output is the same as Listing 46;
- in Listing 51 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 46;
- in Listing 52 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 38.

5.5.1 lookForAlignDelims: spacesBeforeAmpersand

U: 2021-06-19

The `spacesBeforeAmpersand` can be specified in a few different ways. The *basic* form is demonstrated in Listing 40, but we can customise the behaviour further by specifying if we would like this value to change if it encounters a *leading blank column*; that is, when the first column contains only zero-width entries. We refer to this as the *advanced* form.

We demonstrate this feature in relation to Listing 53; upon running the following command

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

then we receive the default output given in Listing 54.

LISTING 53: aligned1.tex

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

LISTING 54: aligned1-default.tex

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

The settings in Listings 55 to 58 are all equivlanent; we have used the not-yet discussed `noAdditionalIndent` field (see Section 5.8 on page 48) which will assist in the demonstration in what follows.

LISTING 55: sba1.yaml

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

LISTING 56: sba2.yaml

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

LISTING 57: sba3.yaml

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

LISTING 58: sba4.yaml

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

Upon running the following commands

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

then we receive the (same) output given in Listing 59; we note that there is *one space* before each ampersand.

LISTING 59: aligned1-mod1.tex

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

We note in particular:

- Listing 55 demonstrates the *basic* form for `lookForAlignDelims`; in this case, the default values are specified as in Listing 36 on page 31;
- Listing 56 demonstrates the *advanced* form for `lookForAlignDelims` and specified `spacesBeforeAmpersand`. The default value is `1`;
- Listing 57 demonstrates the new *advanced* way to specify `spacesBeforeAmpersand`, and for us to set the `default` value that sets the number of spaces before ampersands which are *not* in leading blank columns. The default value is `1`.

  We note that `leadingBlankColumn` has not been specified in Listing 57, and it will inherit the value from `default`;
- Listing 58 demonstrates spaces to be used before amperands for *leading blank columns*. We note that *default* has not been specified, and it will be set to `1` by default.

We can customise the space before the ampersand in the *leading blank column* of Listing 59 by using either of Listings 60 and 61, which are equivalent.

LISTING 60: sba5.yaml

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

LISTING 61: sba6.yaml

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

Upon running

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

then we receive the (same) output given in Listing 62. We note that the space before the ampersand in the *leading blank column* has been set to 0 by Listing 61.

We can demonstrated this feature further using the settings in Listing 64 which give the output in Listing 63.

LISTING 62: aligned1-mod5.tex

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

LISTING 63: aligned1.tex using Listing 64

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

LISTING 64: sba7.yaml

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

5.5.2 lookForAlignDelims: alignFinalDoubleBackSlash

N: 2020-03-21

We explore the `alignFinalDoubleBackSlash` feature by using the file in Listing 65. Upon running the following commands

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

then we receive the respective outputs given in Listing 66 and Listing 67.

LISTING 65: tabular4.tex

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

LISTING 66: tabular4-default.tex

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

LISTING 67: tabular4-FDBS.tex

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

We note that in:

- Listing 66, by default, the *first* set of double back slashes in the first row of the `tabular` environment have been used for alignment;
- Listing 67, the *final* set of double back slashes in the first row have been used, because we specified `alignFinalDoubleBackSlash` as 1.

As of Version 3.0, the alignment routine works on mandatory and optional arguments within commands, and also within 'special' code blocks (see `specialBeginEnd` on page 42); 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 68 and 69 are achievable by default.

LISTING 68: matrix1.tex

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

LISTING 69: 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 70; the default output is shown in Listing 71. 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 70: align-block.tex

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

LISTING 71: align-block.tex default output

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

With reference to Table 1 on page 49 and the, yet undiscussed, fields of noAdditionalIndent and indentRules (see Section 5.8 on page 48), these comment-marked blocks are considered environments.

5.5.3 lookForAlignDelims: the dontMeasure feature

N: 2020-03-21

The lookForAlignDelims field can, optionally, receive the dontMeasure option which can be specified in a few different ways. We will explore this feature in relation to the code given in Listing 72; the default output is shown in Listing 73.

LISTING 72: tabular-DM.tex

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

LISTING 73: tabular-DM.tex default output

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

The dontMeasure field can be specified as largest, and in which case, the largest element will not be measured; with reference to the YAML file given in Listing 75, we can run the command

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

and receive the output given in Listing 74.

LISTING 74: tabular-DM.tex using Listing 75

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

LISTING 75: dontMeasure1.yaml

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

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

The dontMeasure field can also be specified in the form demonstrated in Listing 77. On running the following commands,

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

we receive the output in Listing 76.

LISTING 76: tabular-DM.tex using Listing 77 or Listing 79

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

LISTING 77: dontMeasure2.yaml

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

We note that in Listing 77 we have specified entries not to be measured, one entry per line.

The dontMeasure field can also be specified in the forms demonstrated in Listing 79 and Listing 80. Upon running the commands

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

we receive the output given in Listing 78

LISTING 78: tabular-DM.tex using Listing 79 or Listing 79

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

LISTING 79: dontMeasure3.yaml

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

LISTING 80: dontMeasure4.yaml

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

We note that in:

- Listing 79 we have specified entries not to be measured, each one has a *string* in the this field, together with an optional specification of applyTo as cell;
- Listing 80 we have specified entries not to be measured as a *regular expression* using the regex field, together with an optional specification of applyTo as cell field, together with an optional specification of applyTo as cell.

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

We may also specify the applyTo field as row, a demonstration of which is given in Listing 82; upon running

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

we receive the output in Listing 81.

LISTING 81: tabular-DM.tex using Listing 82

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

LISTING 82: dontMeasure5.yaml

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

Finally, the `applyTo` field can be specified as `row`, together with a `regex` expression. For example, for the settings given in Listing 84, upon running

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

we receive the output in Listing 83.

LISTING 83: tabular-DM.tex using Listing 84

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

LISTING 84: dontMeasure6.yaml

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

5.5.4 lookForAlignDelims: the delimiterRegEx and delimiterJustification feature

N: 2020-03-21

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

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

We demonstrate how to customise this with respect to the code given in Listing 85; the default output from `latexindent.pl` is given in Listing 86.

LISTING 85: tabbing.tex

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

LISTING 86: tabbing.tex default output

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

Let's say that we wish to align the code at either the `\=` or `\>`. We employ the settings given in Listing 88 and run the command

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

to receive the output given in Listing 87.

LISTING 87: tabbing.tex using Listing 88

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

LISTING 88: delimiterRegEx1.yaml

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

We note that:

- in Listing 87 the code has been aligned, as intended, at both the \= and \>;
- in Listing 88 we have heeded the warning and captured the expression using grouping parenthesis, specified a backslash using \\ and said that it must be followed by either = or >.

We can explore delimiterRegEx a little further using the settings in Listing 90 and run the command

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

to receive the output given in Listing 89.

LISTING 89: tabbing.tex using Listing 90

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

LISTING 90: delimiterRegEx2.yaml

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

We note that only the \> have been aligned.

Of course, the other lookForAlignDelims options can be used alongside the delimiterRegEx; regardless of the type of delimiter being used (ampersand or anything else), the fields from Listing 36 on page 31 remain the same; for example, using the settings in Listing 92, and running

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

to receive the output given in Listing 91.

LISTING 91: tabbing.tex using Listing 92

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

LISTING 92: delimiterRegEx3.yaml

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

It is possible that delimiters specified within delimiterRegEx can be of different lengths. Consider the file in Listing 93, and associated YAML in Listing 95. Note that the Listing 95 specifies the option for the delimiter to be either # or \>, *which are different lengths*. Upon running the command

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

we receive the output in Listing 94.

LISTING 93: tabbing1.tex

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

LISTING 94: tabbing1-mod4.tex

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

LISTING 95: delimiterRegEx4.yaml

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

You can set the *delimiter* justification as either `left` (default) or `right`, which will only have effect when delimiters in the same column have different lengths. Using the settings in Listing 97 and running the command

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

gives the output in Listing 96.

LISTING 96: tabbing1-mod5.tex

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

LISTING 97: delimiterRegEx5.yaml

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

Note that in Listing 96 the second set of delimiters have been *right aligned* – it is quite subtle!

5.6 Indent after items, specials and headings

`indentAfterItems`: ⟨*fields*⟩

The environment names specified in `indentAfterItems` tell `latexindent.pl` to look for `\item` commands; if these switches are set to `1` then indentation will be performed so as indent the code after each `item`. A demonstration is given in Listings 99 and 100

LISTING 98: indentAfterItems

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

LISTING 99: 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 100: 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 21 for details of how to configure user settings, and Listing 13 on page 22 in particular .)

LISTING 101: 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 102 shows the default settings of `specialBeginEnd`.

LISTING 102: `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 22); 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 103 and 104.

LISTING 103: `special1.tex` before

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

LISTING 104: `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 105.

LISTING 105: `specialLR.tex`

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

Now consider the YAML files shown in Listings 106 and 107

LISTING 106: specialsLeftRight.yaml

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

LISTING 107: 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 108 and 109.

LISTING 108: specialLR.tex using Listing 106

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

LISTING 109: specialLR.tex using Listings 106 and 107

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

Notice that in:

- Listing 108 the `\left` has been treated as a *command*, with one optional argument;
- Listing 109 the `specialBeginEnd` pattern in Listing 106 has been obeyed because Listing 107 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 110.

LISTING 110: 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 111 and 113 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 112 and 114.

LISTING 111: middle.yaml

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

LISTING 112: special2.tex using Listing 111

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

LISTING 113: middle1.yaml

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

LISTING 114: special2.tex using Listing 113

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

We note that:

- in Listing 112 the bodies of each of the `Elsif` statements have been indented appropriately;
- the `Else` statement has *not* been indented appropriately in Listing 112 – read on!
- we have specified multiple settings for the `middle` field using the syntax demonstrated in Listing 113 so that the body of the `Else` statement has been indented appropriately in Listing 114.

N: 2018-08-13

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

For example, beginning with the code in Listing 116 and the YAML in Listing 115, and running

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

then the output in Listing 116 is unchanged.

LISTING 115: special-verb1.yaml

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

LISTING 116: special3.tex and output using Listing 115

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

We can combine the `specialBeginEnd` with the `lookForAlignDelims` feature. We begin with the code in Listing 117.

LISTING 117: special-align.tex

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

Let's assume that our goal is to align the code at the edge and node text; we employ the code given in Listing 118 and run the command

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

to receive the output in Listing 119.

LISTING 118: edge-node1.yaml

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

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

LISTING 119: special-align.tex using Listing 118

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

The output in Listing 119 is not quite ideal. We can tweak the settings within Listing 118 in order to improve the output; in particular, we employ the code in Listing 120 and run the command

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

to receive the output in Listing 121.

LISTING 120: edge-node2.yaml

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

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

LISTING 121: special-align.tex using Listing 120

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

U: 2021-06-19

The lookForThis field can be considered optional; by default, it is assumed to be 1, which is demonstrated in Listing 120.

`indentAfterHeadings`: ⟨*fields*⟩

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

LISTING 122: `indentAfterHeadings`

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

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

You can add any of your own custom heading commands to this field, specifying the `level` as appropriate. You can also specify your own indentation in `indentRules` (see Section 5.8 on page 48); 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 123 saved into `headings1.yaml`, and that you have the text from Listing 124 saved into `headings1.tex`.

LISTING 123: `headings1.yaml`

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

LISTING 124: `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 125.

[5] There is a slight difference in interface for this field when comparing Version 2.2 to Version 3.0; see appendix F on page 140 for details.

LISTING 125: headings1.tex using Listing 123

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

LISTING 126: 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 123 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 126; 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 [25], and is *off* by default.

For example, consider the example shown in Listing 127 together with the default output shown in Listing 128.

LISTING 127: mult-nested.tex

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

LISTING 128: 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 129 and that you run the following command:

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

You should receive the output shown in Listing 130.

LISTING 129: max-indentation1.yaml

```
maximumIndentation: " "
```

LISTING 130: mult-nested.tex using Listing 129

```
\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 128 and 130 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 18 on page 27) or `noIndentBlock` (see Listing 20 on page 27).

5.7 The code blocks known latexindent.pl

As of Version 3.0, `latexindent.pl` processes documents using code blocks; each of these are shown in Table 1.

N: 2019-07-13

We will refer to these code blocks in what follows. Note that the fine tuning of the definition of the code blocks detailed in Table 1 is discussed in Section 8 on page 124.

5.8 noAdditionalIndent and indentRules

`latexindent.pl` operates on files by looking for code blocks, as detailed in Section 5.7; for each type of code block in Table 1 on the following page (which we will call a ⟨*thing*⟩ in what follows) it searches YAML fields for information in the following order:

1. `noAdditionalIndent` for the *name* of the current ⟨*thing*⟩;
2. `indentRules` for the *name* of the current ⟨*thing*⟩;
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 1 on the next page; for reference, there follows a list of the code blocks covered.

5.8.1 Environments and their arguments 50
5.8.2 Environments with items 56
5.8.3 Commands with arguments 57
5.8.4 ifelsefi code blocks 59
5.8.5 specialBeginEnd code blocks 61
5.8.6 afterHeading code blocks 62
5.8.7 The remaining code blocks 64
keyEqualsValuesBracesBrackets 64

TABLE 1: 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 98 and 101 on page 41	`\begin{enumerate}`
		`  \item ...`
		`\end{enumerate}`

specialBeginEnd	User specified, see Listing 102 on page 42	`\[`
		`  ...`
		`\]`

afterHeading	User specified, see Listing 122 on page 46	`\chapter{title}`
		`  ...`
		`\section{title}`

filecontents	User specified, see Listing 28 on page 29	`\begin{filecontents}`
		`...`
		`\end{filecontents}`

namedGroupingBracesBrackets 65
UnNamedGroupingBracesBrackets 65
filecontents 66
5.8.8 Summary 66

5.8.1 Environments and their arguments

There are a few different YAML switches governing the indentation of environments; let's start with the code shown in Listing 131.

LISTING 131: 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 132 and 133.

LISTING 132: myenv-noAdd1.yaml

```
noAdditionalIndent:
    myenv: 1
```

LISTING 133: 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 134; note in particular that the environment `myenv` has not received any *additional* indentation, but that the `outer` environment *has* still received indentation.

LISTING 134: myenv.tex output (using either Listing 132 or Listing 133)

```
\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 135 and 136, 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 137.

LISTING 135: myenv-noAdd3.yaml

```
noAdditionalIndent:
    myenv: 0
```

LISTING 136: myenv-noAdd4.yaml

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

LISTING 137: myenv.tex output (using either Listing 135 or Listing 136)

```
\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 138.

LISTING 138: 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 139; 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 132), then *all* parts of the environment (body, optional and mandatory arguments) are assumed to want no additional indent.

LISTING 139: myenv-args.tex using Listing 132

```
\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 140 and 141.

LISTING 140: myenv-noAdd5.yaml

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

LISTING 141: 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 142 and 143. Note that in Listing 142 the text for the *optional* argument has not received any additional indentation, and that in Listing 143 the *mandatory* argument has not received any additional indentation; in both cases, the *body* has not received any additional indentation.

LISTING 142: myenv-args.tex using Listing 140

```
\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 143: myenv-args.tex using Listing 141

```
\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 144 and 145.

LISTING 144: myenv-rules1.yaml

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

LISTING 145: 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 146; note in particular that the environment `myenv` has received one tab (from the `outer` environment) plus three spaces from Listing 144 or 145.

LISTING 146: myenv.tex output (using either Listing 144 or Listing 145)

```
\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 138 that contains optional and mandatory arguments. Upon using Listing 144 as in

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

we obtain the output in Listing 147; note that the body, optional argument and mandatory argument of `myenv` have *all* received the same customised indentation.

LISTING 147: myenv-args.tex using Listing 144

```
\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 148 and 149

LISTING 148: myenv-rules3.yaml

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

LISTING 149: 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 150 and 151.

LISTING 150: myenv-args.tex using Listing 148

```
\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 151: myenv-args.tex using Listing 149

```
\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 150, 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 151, 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 152). Let's say that you change the value of `environments` to 1 in Listing 152, and that you run

LISTING 152: noAdditionalIndentGlobal

```
322 noAdditionalIndentGlobal:
323     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 153 and 154; in Listing 153 notice that *both* environments receive no additional indentation but that the arguments of `myenv` still *do* receive indentation. In Listing 154 notice that the *outer* environment does not receive additional indentation, but because of the settings from `myenv-rules1.yaml` (in Listing 144 on page 52), the `myenv` environment still *does* receive indentation.

LISTING 153: myenv-args.tex using Listing 152

```
\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 154: myenv-args.tex using Listings 144 and 152

```
\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 155 and 156

LISTING 155: opt-args-no-add-glob.yaml

```
noAdditionalIndentGlobal:
    optionalArguments: 1
```

LISTING 156: 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 157 and 158. Notice that in Listing 157 the *optional* argument has not received any additional indentation, and in Listing 158 the *mandatory* argument has not received any additional indentation.

LISTING 157: myenv-args.tex using Listing 155

```
\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 158: myenv-args.tex using Listing 156

```
\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 159; if you change the `environments` field to anything involving horizontal space, say `" "`, and then run the following commands

LISTING 159: indentRulesGlobal

```
338 indentRulesGlobal:
339     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 160 and 161. Note that in Listing 160, both the environment blocks have received a single-space indentation, whereas in Listing 161 the `outer` environment has received single-space indentation (specified by `indentRulesGlobal`), but `myenv` has received `"  "`, as specified by the particular `indentRules` for `myenv` Listing 144 on page 52.

LISTING 160: myenv-args.tex using Listing 159

```
\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 161: myenv-args.tex using Listings 144 and 159

```
\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 162 and 163

LISTING 162: opt-args-indent-rules-glob.yaml

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

LISTING 163: 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 164 and 165. Note that the *optional* argument in Listing 164 has received two tabs worth of indentation, while the *mandatory* argument has done so in Listing 165.

LISTING 164: myenv-args.tex using Listing 162

```
\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 165: myenv-args.tex using Listing 163

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

5.8.2 Environments with items

With reference to Listings 98 and 101 on page 41, some commands may contain `item` commands; for the purposes of this discussion, we will use the code from Listing 99 on page 41.

Assuming that you've populated `itemNames` with the name of your `item`, you can put the item name into `noAdditionalIndent` as in Listing 166, 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 167

LISTING 166: item-noAdd1.yaml

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

LISTING 167: 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 168 and 169; note that in Listing 168 that the text after each `item` has not received any additional indentation, and in Listing 169, the text after each `item` has received a single space of indentation, specified by Listing 167.

LISTING 168: `items1.tex` using Listing 166

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

LISTING 169: `items1.tex` using Listing 167

```
\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 170 and 171. Note that there is a need to 'reset/remove' the `item` field from `indentRules` in both cases (see the hierarchy description given on page 48) as the `item` command is a member of `indentRules` by default.

LISTING 170: `items-noAdditionalGlobal.yaml`

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

LISTING 171: `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 168 and 169 are obtained; note, however, that *all* such `item` commands without their own individual `noAdditionalIndent` or `indentRules` settings would behave as in these listings.

5.8.3 Commands with arguments

Let's begin with the simple example in Listing 172; when `latexindent.pl` operates on this file, the default output is shown in Listing 173. [6]

LISTING 172: `mycommand.tex`

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

LISTING 173: `mycommand.tex` default output

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

[6] The command code blocks have quite a few subtleties, described in Section 5.9 on page 66.

As in the environment-based case (see Listings 132 and 133 on page 50) we may specify `noAdditionalIndent` either in 'scalar' form, or in 'field' form, as shown in Listings 174 and 175

LISTING 174: mycommand-noAdd1.yaml

```
noAdditionalIndent:
    mycommand: 1
```

LISTING 175: 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 176 and 177

LISTING 176: mycommand.tex using Listing 174

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

LISTING 177: mycommand.tex using Listing 175

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

Note that in Listing 176 that the 'body', optional argument *and* mandatory argument have *all* received no additional indentation, while in Listing 177, 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 140 and 141 on page 52; explicit examples are given in Listings 178 and 179.

LISTING 178: mycommand-noAdd3.yaml

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

LISTING 179: 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 180 and 181.

LISTING 180: mycommand.tex using Listing 178

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

LISTING 181: mycommand.tex using Listing 179

```
\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 180 and 181 has received no additional indent, even though `body` is explicitly set to `0` in both Listings 178 and 179. This is because, by default, `noAdditionalIndentGlobal` for `commands` is set to `1` by default; this can be easily fixed as in Listings 182 and 183.

LISTING 182: mycommand-noAdd5.yaml

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

LISTING 183: 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 184 and 185.

LISTING 184: mycommand.tex using Listing 182

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

LISTING 185: mycommand.tex using Listing 183

```
\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 148 and 149 on page 53 and Listings 159, 162 and 163 on pages 55–56.

5.8.4 ifelsefi code blocks

Let's use the simple example shown in Listing 186; when `latexindent.pl` operates on this file, the output as in Listing 187; note that the body of each of the `\if` statements have been indented, and that the `\else` statement has been accounted for correctly.

LISTING 186: ifelsefi1.tex

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

LISTING 187: 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 188 and 189.

LISTING 188: ifnum-noAdd.yaml

```
noAdditionalIndent:
    ifnum: 1
```

LISTING 189: 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 190 and 191; note that in Listing 190, the `ifnum` code block has *not* received any additional indentation, while in Listing 191, the `ifnum` code block has received one tab and two spaces of indentation.

LISTING 190: ifelsefi1.tex using Listing 188

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

LISTING 191: ifelsefi1.tex using Listing 189

```
\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 192 and 193.

LISTING 192: ifelsefi-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
    ifElseFi: 1
```

LISTING 193: 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 194 and 195; notice that in Listing 194 neither of the `ifelsefi` code blocks have received indentation, while in Listing 195 both code blocks have received a single space of indentation.

LISTING 194: ifelsefi1.tex using Listing 192

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

LISTING 195: ifelsefi1.tex using Listing 193

```
\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 196, and the associated default output given in Listing 197; note, in particular, that the bodies of each of the 'or statements' have been indented.

LISTING 196: ifelsefi2.tex

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

LISTING 197: ifelsefi2.tex default output

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

5.8.5 specialBeginEnd code blocks

Let's use the example from Listing 103 on page 42 which has default output shown in Listing 104 on page 42.

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 198 and 199.

LISTING 198: displayMath-noAdd.yaml

```
noAdditionalIndent:
    displayMath: 1
```

LISTING 199: 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 200 and 201; note that in Listing 200, the `displayMath` code block has *not* received any additional indentation, while in Listing 201, the `displayMath` code block has received three tabs worth of indentation.

LISTING 200: special1.tex using Listing 198

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

LISTING 201: special1.tex using Listing 199

```
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 202 and 203.

LISTING 202: special-noAdd-glob.yaml

```
noAdditionalIndentGlobal:
    specialBeginEnd: 1
```

LISTING 203: 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 204 and 205; notice that in Listing 204 neither of the `special` code blocks have received indentation, while in Listing 205 both code blocks have received a single space of indentation.

LISTING 204: special1.tex using Listing 202

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

LISTING 205: special1.tex using Listing 203

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

5.8.6 afterHeading code blocks

Let's use the example Listing 206 for demonstration throughout this Section. As discussed on page 46, by default `latexindent.pl` will not add indentation after headings.

LISTING 206: headings2.tex

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

On using the YAML file in Listing 208 by running the command

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

we obtain the output in Listing 207. Note that the argument of `paragraph` has received (default) indentation, and that the body after the heading statement has received (default) indentation.

LISTING 207: headings2.tex using Listing 208

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

LISTING 208: headings3.yaml

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

If we specify `noAdditionalIndent` as in Listing 210 and run the command

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

then we receive the output in Listing 209. 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 209: headings2.tex using Listing 210

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

LISTING 210: headings4.yaml

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

Similarly, if we specify `indentRules` as in Listing 212 and run analogous commands to those above, we receive the output in Listing 211; note that the *body*, *mandatory argument* and content *after the heading* of `paragraph` have *all* received three tabs worth of indentation.

LISTING 211: headings2.tex using Listing 212

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

LISTING 212: headings5.yaml

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

We may, instead, specify `noAdditionalIndent` in 'field' form, as in Listing 214 which gives the output in Listing 213.

LISTING 213: headings2.tex using Listing 214

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

LISTING 214: headings6.yaml

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

Analogously, we may specify `indentRules` as in Listing 216 which gives the output in Listing 215; 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 215: headings2.tex using Listing 216

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

LISTING 216: 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 218 and 220 respectively, with respective output in Listings 217 and 219. Note that in Listing 218 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 219, the *argument* has received both a (default) tab plus two spaces of indentation (from the global rule specified in Listing 220), and the remaining body after `paragraph` has received just two spaces of indentation.

LISTING 217: headings2.tex using Listing 218

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

LISTING 218: headings8.yaml

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

LISTING 219: headings2.tex using Listing 220

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

LISTING 220: headings9.yaml

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

5.8.7 The remaining code blocks

Referencing the different types of code blocks in Table 1 on page 49, we have a few code blocks yet to cover; these are very similar to the `commands` code block type covered comprehensively in Section 5.8.3 on page 57, 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 1 on page 49;
- then an = symbol;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

N: 2019-07-13

See the `keyEqualsValuesBracesBrackets: follow` and `keyEqualsValuesBracesBrackets: name` fields of the fine tuning section in Listing 497 on page 124

An example is shown in Listing 221, with the default output given in Listing 222.

LISTING 221: pgfkeys1.tex

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

LISTING 222: pgfkeys1.tex default output

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

In Listing 222, note that the maximum indentation is three tabs, and these come from:

- the `\pgfkeys` command's mandatory argument;
- the `start coordinate/.initial` key's mandatory argument;
- the `start coordinate/.initial` key's body, which is defined as any lines following the name of the key that include its arguments. This is the part controlled by the *body* field for `noAdditionalIndent` and friends from page 48.

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 1 on page 49;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

N: 2019-07-13

See the `NamedGroupingBracesBrackets: follow` and `NamedGroupingBracesBrackets: name` fields of the fine tuning section in Listing 497 on page 124

A simple example is given in Listing 223, with default output in Listing 224.

LISTING 223: `child1.tex`

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

LISTING 224: `child1.tex` default output

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

In particular, `latexindent.pl` considers `child`, `parent` and `node` all to be `namedGroupingBracesBrackets`[7]. Referencing Listing 224, note that the maximum indentation is two tabs, and these come from:

- the `child`'s mandatory argument;
- the `child`'s body, which is defined as any lines following the name of the `namedGroupingBracesBrackets` that include its arguments. This is the part controlled by the *body* field for `noAdditionalIndent` and friends from page 48.

UnNamedGroupingBracesBrackets occur in a variety of situations; specifically, we define this type of code block as satisfying the following criteria:

- it must immediately follow either `{` OR `[` OR `,` OR `&` OR `)` OR `(` OR `$`;
- then at least one set of curly braces or square brackets (comments and line breaks allowed throughout).

N: 2019-07-13

See the `UnNamedGroupingBracesBrackets: follow` field of the fine tuning section in Listing 497 on page 124

An example is shown in Listing 225 with default output give in Listing 226.

LISTING 225: `psforeach1.tex`

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

LISTING 226: `psforeach1.tex` default output

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

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

Referencing Listing 226, there are *three* sets of unnamed braces. Note also that the maximum value of indentation is three tabs, and these come from:

- the `\psforeach` command's mandatory argument;
- the *first* un-named braces mandatory argument;
- the *first* un-named braces *body*, which we define as any lines following the first opening `{` or `[` that defined the code block. This is the part controlled by the *body* field for `noAdditionalIndent` and friends from page 48.

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.

5.8.8 Summary

Having considered all of the different types of code blocks, the functions of the fields given in Listings 227 and 228 should now make sense.

LISTING 227: `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 228: `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
```

5.9 Commands and the strings between their arguments

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

`commandCodeBlocks`: ⟨*fields*⟩

U: 2018-04-27

The `commandCodeBlocks` field contains a few switches detailed in Listing 229.

LISTING 229: commandCodeBlocks

```
353 commandCodeBlocks:
354     roundParenthesesAllowed: 1
355     stringsAllowedBetweenArguments:
356       -
357         amalgamate: 1
358       - 'node'
359       - 'at'
360       - 'to'
361       - 'decoration'
362       - '\+\+'
363       - '\-\-'
364       - '\#\#\d'
365     commandNameSpecial:
366       -
367         amalgamate: 1
368       - '@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 230.

LISTING 230: pstricks1.tex

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

LISTING 231: 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 229, then `latexindent.pl` will allow round parenthesis between optional and mandatory arguments. In the case of the code in Listing 230, `latexindent.pl` finds *all* the arguments of `defFunction`, both before and after `(u,v)`.

The default output from running `latexindent.pl` on Listing 230 actually leaves it unchanged (see Listing 231); note in particular, this is because of `noAdditionalIndentGlobal` as discussed on page 59.

Upon using the YAML settings in Listing 233, and running the command

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

we obtain the output given in Listing 232.

LISTING 232: pstricks1.tex using Listing 233

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

LISTING 233: noRoundParentheses.yaml

```
commandCodeBlocks:
    roundParenthesesAllowed: 0
```

Notice the difference between Listing 231 and Listing 232; in particular, in Listing 232, 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 1 on page 49) which, by default, assume indentation for their body, and hence the tabbed indentation in Listing 232.

Let's explore this using the YAML given in Listing 235 and run the command

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

then the output is as in Listing 234.

LISTING 234: pstricks1.tex using Listing 235

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

LISTING 235: defFunction.yaml

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

Notice in Listing 234 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 235.

`stringsAllowedBetweenArguments`: ⟨*fields*⟩

`tikz` users may well specify code such as that given in Listing 236; processing this code using `latexindent.pl` gives the default output in Listing 237.

LISTING 236: tikz-node1.tex

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

LISTING 237: 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 229 on the preceding page, we see that the strings

to, node, ++

are all allowed to appear between arguments; importantly, you are encouraged to add further names to this field as necessary. This means that when `latexindent.pl` processes Listing 236, 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 239 and running the command

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

we receive the output given in Listing 238.

LISTING 238: tikz-node1.tex using Listing 239

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

LISTING 239: draw.yaml

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

Notice that each line after the `\draw` command (its 'body') in Listing 238 has been given the appropriate two-spaces worth of indentation specified in Listing 239.

Let's compare this with the output from using the YAML settings in Listing 241, and running the command

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

given in Listing 240.

LISTING 240: tikz-node1.tex using Listing 241

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

LISTING 241: 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 1 on page 49) *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 229 on page 67,, 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 242 orListing 243 is equivalent to using the settings in Listing 244.

LISTING 242: amalgamate-demo.yaml

```
commandCodeBlocks:

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

LISTING 243: amalgamate-demo1.yaml

```
commandCodeBlocks:

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

LISTING 244: 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 245 means that only the strings specified in that field will be used.

LISTING 245: 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 243 to 245.

We may explore this feature further with the code in Listing 246, whose default output is given in Listing 247.

LISTING 246: for-each.tex

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

LISTING 247: 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 249, and running the command

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

given in Listing 248.

LISTING 248: for-each.tex using Listing 249

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

LISTING 249: foreach.yaml

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

You might like to compare the output given in Listing 247 and Listing 248. Note,in particular, in Listing 247 that the `foreach` command has not included any of the subsequent strings, and that the braces have been treated as a `namedGroupingBracesBrackets`. In Listing 248 the `foreach` command has been allowed to have `\x/\y` and `in` between arguments because of the settings given in Listing 249.

`commandNameSpecial`: ⟨*fields*⟩

U: 2018-04-27

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

For example, consider the sample file in Listing 250, which has default output in Listing 251.

LISTING 250: ifnextchar.tex

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

LISTING 251: ifnextchar.tex default output

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

Notice that in Listing 251 the `parbox` command has been able to indent its body, because `latexindent.pl` has successfully found the command `\@ifnextchar` first; the pattern-matching of `latexindent.pl` starts from *the inner most <thing> and works outwards*, discussed in more detail on page 113.

For demonstration, we can compare this output with that given in Listing 252 in which the settings from Listing 253 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 252: `ifnextchar.tex` using Listing 253

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

LISTING 253: `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 243 to 245.

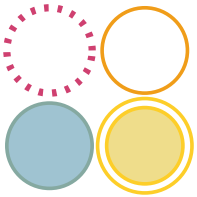
SECTION 6

The -m (modifylinebreaks) switch

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

6.1 textWrapOptions: modifying line breaks by text wrapping . . . 73
6.1.1 text wrapping on a per-code-block basis . . . 76
6.1.2 Summary of text wrapping . . . 82
6.2 oneSentencePerLine: modifying line breaks for sentences . . . 82
6.2.1 sentencesFollow . . . 84
6.2.2 sentencesBeginWith . . . 85
6.2.3 sentencesEndWith . . . 86
6.2.4 Features of the oneSentencePerLine routine . . . 88
6.2.5 text wrapping and indenting sentences . . . 89
6.3 removeParagraphLineBreaks: modifying line breaks for paragraphs . . . 91
6.4 Combining removeParagraphLineBreaks and textWrapOptions . . . 97
6.5 Poly-switches . . . 98
6.6 modifyLineBreaks for environments . . . 99
6.6.1 Adding line breaks: BeginStartsOnOwnLine and BodyStartsOnOwnLine . . . 99
6.6.2 Adding line breaks using EndStartsOnOwnLine and EndFinishesWithLineBreak 101
6.6.3 poly-switches 1, 2, and 3 only add line breaks when necessary . . . 102
6.6.4 Removing line breaks (poly-switches set to −1) . . . 103
6.6.5 About trailing horizontal space . . . 104
6.6.6 poly-switch line break removal and blank lines . . . 105
6.7 Poly-switches for double back slash . . . 106
6.7.1 Double back slash starts on own line . . . 106
6.7.2 Double back slash finishes with line break . . . 107
6.7.3 Double back slash poly-switches for specialBeginEnd . . . 107
6.7.4 Double back slash poly-switches for optional and mandatory arguments . . . 108
6.7.5 Double back slash optional square brackets . . . 109
6.8 Poly-switches for other code blocks . . . 109

`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 254.

LISTING 254: `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 255 shows a sample file with blank lines; upon running

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

the output is shown in Listing 256; note that the multiple blank lines have been condensed into one blank line, and note also that we have used the `-m` switch!

LISTING 255: `mlb1.tex`

```
before blank line



after blank line


after blank line
```

LISTING 256: `mlb1.tex` out output

```
before blank line

after blank line

after blank line
```

6.1 textWrapOptions: modifying line breaks by text wrapping

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 257. 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 257: `textWrapOptions` (`-m`)

```
    textWrapOptions:
        columns: 0
```

For example, consider the file give in Listing 258.

LISTING 258: 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 260, and running the command

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

we obtain the output in Listing 259.

LISTING 259: 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 260: 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 261,

LISTING 261: 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 260,

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

then the output is as in Listing 262.

LISTING 262: 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 263

LISTING 263: 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 260,

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

then the output is as in Listing 264.

LISTING 264: 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 [26]). By default, the separator is empty which means that a new line token will be used, but you can change it as you see fit.

For example starting with the file in Listing 265

LISTING 265: textwrap4.tex

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

and using `textwrap2.yaml` from Listing 267 with the following command

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

then we obtain the output in Listing 266.

LISTING 266: textwrap4-mod2.tex

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

LISTING 267: textwrap2.yaml -m

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

N: 2019-09-07

There are options to specify the `huge` option for the `Text::Wrap` module [26] . This can be helpful if you would like to forbid the `Text::Wrap` routine from breaking words. For example, using the settings in Listings 269 and 271 and running the commands

```
cmh:~$ latexindent.pl -m textwrap4.tex -o=+-mod2A -l textwrap2A.yaml
cmh:~$ latexindent.pl -m textwrap4.tex -o=+-mod2B -l textwrap2B.yaml
```

gives the respective output in Listings 268 and 270.

LISTING 268: textwrap4-mod2A.tex

```
He
re
is
a
li
ne
of
te
xt
.
```

LISTING 269: textwrap2A.yaml -m

```
modifyLineBreaks:
  textWrapOptions:
    columns: 3
```

LISTING 270: textwrap4-mod2B.tex

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

LISTING 271: textwrap2B.yaml -m

```
modifyLineBreaks:
  textWrapOptions:
    columns: 3
    huge: overflow
```

N: 2020-11-06

You can also specify the `tabstop` field as an integer value, which is passed to the text wrap module; see [26] for details. Starting with the code in Listing 272 with settings in Listing 273, and running the command

```
cmh:~$ latexindent.pl -m textwrap-ts.tex -o=+-mod1 -l tabstop.yaml
```

gives the code given in Listing 274.

LISTING 272: textwrap-ts.tex

```
x␣␣␣␣␣␣␣␣y
```

LISTING 273: tabstop.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 80
        tabstop: 9
```

LISTING 274: textwrap-ts-mod1.tex

```
x␣␣␣␣␣␣␣␣y
```

You can specify `break` and `unexpand` options in your settings in analogous ways to those demonstrated in Listings 271 and 273, and they will be passed to the `Text::Wrap` module. I have not found a useful reason to do this; see [26] for more details.

6.1.1 text wrapping on a per-code-block basis

U: 2018-08-13

By default, if the value of `columns` is greater than 0 and the `-m` switch is active, then the text wrapping routine will operate before the code blocks have been searched for. This behaviour is customisable;

in particular, you can instead instruct `latexindent.pl` to apply `textWrap` on a per-code-block basis. Thanks to [30] for their help in testing and shaping this feature.

The full details of `textWrapOptions` are shown in Listing 275. In particular, note the field `perCodeBlockBasis: 0`.

LISTING 275: `textWrapOptions` -m

```
510     textWrapOptions:
511         columns: 0
512         # huge: overflow    # forbid mid-word line breaks
513         separator: ""
514         perCodeBlockBasis: 0
515         all: 0
516         alignAtAmpersandTakesPriority: 1
517         environments:
518             quotation: 0
519         ifElseFi: 0
520         optionalArguments: 0
521         mandatoryArguments: 0
522         items: 0
523         specialBeginEnd: 0
524         afterHeading: 0
525         preamble: 0
526         filecontents: 0
527         masterDocument: 0
```

The code blocks detailed in Listing 275 are with direct reference to those detailed in Table 1 on page 49. 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. The same notation is used between this feature and the `removeParagraphLineBreaks` described in Listing 336 on page 91; in fact, the two features can even be combined (this is detailed in Section 6.4 on page 97).

Let's explore these switches with reference to the code given in Listing 276; the text outside of the environment is considered part of the `masterDocument`.

LISTING 276: `textwrap5.tex`

```
Before the environment; here is a line of text that can be wrapped by latexindent.pl.

\begin{myenv}
Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{myenv}

After the environment; here is a line of text that can be wrapped by latexindent.pl.
```

With reference to this code block, the settings given in Listings 277 to 279 each give the same output.

LISTING 277: `textwrap3.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 30
        perCodeBlockBasis: 1
        all: 1
```

LISTING 278: `textwrap4.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 30
        perCodeBlockBasis: 1
        environments: 1
        masterDocument: 1
```

LISTING 279: `textwrap5.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 30
        perCodeBlockBasis: 1
        environments:
          myenv: 1
        masterDocument: 1
```

Let's explore the similarities and differences in the equivalent (with respect to Listing 276) syntax specified in Listings 277 to 279:

- in each of Listings 277 to 279 notice that `columns: 30`;
- in each of Listings 277 to 279 notice that `perCodeBlockBasis: 1`;

- in Listing 277 we have specified `all: 1` so that the text wrapping will operate upon *all* code blocks;
- in Listing 278 we have *not* specified `all`, and instead, have specified that text wrapping should be applied to each of `environments` and `masterDocument`;
- in Listing 279 we have specified text wrapping for `masterDocument` and on a *per-name* basis for `environments` code blocks.

Upon running the following commands

```
cmh:~$ latexindent.pl -s textwrap5.tex -l=textwrap3.yaml -m
cmh:~$ latexindent.pl -s textwrap5.tex -l=textwrap4.yaml -m
cmh:~$ latexindent.pl -s textwrap5.tex -l=textwrap5.yaml -m
```

we obtain the output shown in Listing 280.

LISTING 280: textwrap5-mod3.tex

```
Before the environment; here
is a line of text that can be
wrapped by latexindent.pl.

\begin{myenv}
    Within the environment; here
    is a line of text that can be
    wrapped by latexindent.pl.
\end{myenv}

After the environment; here
is a line of text that can be
wrapped by latexindent.pl.
```

We can explore the idea of per-name text wrapping given in Listing 279 by using Listing 281.

LISTING 281: textwrap6.tex

```
Before the environment; here is a line of text that can be wrapped by latexindent.pl.

\begin{myenv}
Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{myenv}

\begin{another}
Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{another}

After the environment; here is a line of text that can be wrapped by latexindent.pl.
```

In particular, upon running

```
cmh:~$ latexindent.pl -s textwrap6.tex -l=textwrap5.yaml -m
```

we obtain the output given in Listing 282.

LISTING 282: textwrap6.tex using Listing 279

```
Before the environment; here
is a line of text that can be
wrapped by latexindent.pl.

\begin{myenv}
    Within the environment; here
    is a line of text that can be
    wrapped by latexindent.pl.
\end{myenv}

\begin{another}
    Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{another}

After the environment; here
is a line of text that can be
wrapped by latexindent.pl.
```

Notice that, because environments has been specified only for myenv (in Listing 279) that the environment named another has *not* had text wrapping applied to it.

The all field can be specified with exceptions which can either be done on a per-code-block or per-name basis; we explore this in relation to Listing 281 in the settings given in Listings 283 to 285.

LISTING 283: textwrap6.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 30
        perCodeBlockBasis: 1
        all:
          except:
            - environments
```

LISTING 284: textwrap7.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 30
        perCodeBlockBasis: 1
        all:
          except:
            - myenv
```

LISTING 285: textwrap8.yaml -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 30
        perCodeBlockBasis: 1
        all:
          except:
            - masterDocument
```

Upon running the commands

```
cmh:~$ latexindent.pl -s textwrap6.tex -l=textwrap6.yaml -m
cmh:~$ latexindent.pl -s textwrap6.tex -l=textwrap7.yaml -m
cmh:~$ latexindent.pl -s textwrap6.tex -l=textwrap8.yaml -m
```

we receive the respective output given in Listings 286 to 288.

LISTING 286: `textwrap6.tex` using Listing 283

```
Before the environment; here
is a line of text that can be
wrapped by latexindent.pl.

\begin{myenv}
    Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{myenv}

\begin{another}
    Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{another}

After the environment; here
is a line of text that can be
wrapped by latexindent.pl.
```

LISTING 287: `textwrap6.tex` using Listing 284

```
Before the environment; here
is a line of text that can be
wrapped by latexindent.pl.

\begin{myenv}
    Within the environment; here is a line of text that can be wrapped by latexindent.pl.
\end{myenv}

\begin{another}
    Within the environment; here
    is a line of text that can be
    wrapped by latexindent.pl.
\end{another}

After the environment; here
is a line of text that can be
wrapped by latexindent.pl.
```

LISTING 288: `textwrap6.tex` using Listing 285

```
Before the environment; here is a line of text that can be wrapped by latexindent.pl.

\begin{myenv}
    Within the environment; here
    is a line of text that can be
    wrapped by latexindent.pl.
\end{myenv}

\begin{another}
    Within the environment; here
    is a line of text that can be
    wrapped by latexindent.pl.
\end{another}

After the environment; here is a line of text that can be wrapped by latexindent.pl.
```

Notice that:

- in Listing 286 the text wrapping routine has not been applied to any `environments` because it has been switched off (per-code-block) in Listing 283;

- in Listing 287 the text wrapping routine has not been applied to `myenv` because it has been switched off (per-name) in Listing 284;
- in Listing 288 the text wrapping routine has not been applied to `masterDocument` because of the settings in Listing 285.

The `columns` field has a variety of different ways that it can be specified; we've seen two basic ways already: the default (set to `0`) and a positive integer (see Listing 281 on page 78, for example). We explore further options in Listings 289 to 291.

LISTING 289: `textwrap9.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns:
            default: 30
            environments: 50
        perCodeBlockBasis: 1
        all: 1
```

LISTING 290: `textwrap10.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns:
            default: 30
            environments:
                default: 50
        perCodeBlockBasis: 1
        all: 1
```

LISTING 291: `textwrap11.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns:
            default: 30
            environments:
                myenv: 50
                another: 15
        perCodeBlockBasis: 1
        all: 1
```

Listing 289 and Listing 290 are equivalent. Upon running the commands

```
cmh:~$ latexindent.pl -s textwrap6.tex -l=textwrap9.yaml -m
cmh:~$ latexindent.pl -s textwrap6.tex -l=textwrap11.yaml -m
```

we receive the respective output given in Listings 292 and 293.

LISTING 292: `textwrap6.tex` using Listing 289

```
Before the environment; here
is a line of text that can be
wrapped by latexindent.pl.

\begin{myenv}
    Within the environment; here is a line of text
    that can be wrapped by latexindent.pl.
\end{myenv}

\begin{another}
    Within the environment; here is a line of text
    that can be wrapped by latexindent.pl.
\end{another}

After the environment; here
is a line of text that can be
wrapped by latexindent.pl.
```

LISTING 293: `textwrap6.tex` using Listing 291

```
Before the environment; here
is a line of text that can be
wrapped by latexindent.pl.

\begin{myenv}
    Within the environment; here is a line of text
    that can be wrapped by latexindent.pl.
\end{myenv}

\begin{another}
    Within the
    environment;
    here is a line
    of text that
    can be wrapped
    by
    latexindent.pl
    .
\end{another}

After the environment; here
is a line of text that can be
wrapped by latexindent.pl.
```

Notice that:

- in Listing 292 the text for the `masterDocument` has been wrapped using 30 columns, while `environments` has been wrapped using 50 columns;
- in Listing 293 the text for `myenv` has been wrapped using 50 columns, the text for `another` has been wrapped using 15 columns, and `masterDocument` has been wrapped using 30 columns.

If you don't specify a `default` value on per-code-block basis, then the `default` value from `columns` will be inherited; if you don't specify a default value for `columns` then 80 will be used.

`alignAtAmpersandTakesPriority` is set to 1 by default; assuming that text wrapping is occurring on a per-code-block basis, and the current environment/code block is specified within Listing 33 on page 30 then text wrapping will be disabled for this code block.

If you wish to specify `afterHeading` commands (see Listing 122 on page 46) on a per-name basis, then you need to append the name with `:heading`, for example, you might use `section:heading`.

6.1.2 Summary of text wrapping

It is important to note the following:

- Verbatim environments (Listing 18 on page 27) and verbatim commands (Listing 19 on page 27) will *not* be affected by the text wrapping routine (see Listing 262 on page 75);
- comments will *not* be affected by the text wrapping routine (see Listing 264 on page 75);
- it is possible to wrap text on a per-code-block and a per-name basis;
- the text wrapping routine sets `preserveBlankLines` as 1;
- indentation is performed *after* the text wrapping routine; as such, indented code will likely exceed any maximum value set in the `columns` field.

U: 2018-08-13

6.2 oneSentencePerLine: modifying line breaks for sentences

N: 2018-01-13

You can instruct `latexindent.pl` to format your file so that it puts one sentence per line. Thank you to [17] for helping to shape and test this feature. The behaviour of this part of the script is controlled by the switches detailed in Listing 294, all of which we discuss next.

LISTING 294: oneSentencePerLine -m

```
oneSentencePerLine:
    manipulateSentences: 0
    removeSentenceLineBreaks: 1
    textWrapSentences: 0
    sentenceIndent: ""
    sentencesFollow:
        par: 1
        blankLine: 1
        fullStop: 1
        exclamationMark: 1
        questionMark: 1
        rightBrace: 1
        commentOnPreviousLine: 1
        other: 0
    sentencesBeginWith:
        A-Z: 1
        a-z: 0
        other: 0
    sentencesEndWith:
        basicFullStop: 0
        betterFullStop: 1
        exclamationMark: 1
        questionMark: 1
        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 line breaks 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 295.

LISTING 295: 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 297 and 299, 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 296 and 298.

LISTING 296: multiple-sentences.tex using Listing 297

```
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 297: manipulate-sentences.yaml -m

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

LISTING 298: multiple-sentences.tex using Listing 299

```
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 299: keep-sen-line-breaks.yaml -m

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

Notice, in particular, that the 'internal' sentence line breaks in Listing 295 have been removed in Listing 296, but have not been removed in Listing 298.

The remainder of the settings displayed in Listing 294 on the previous page instruct `latexindent.pl` on how to define a sentence. From the perspective of `latexindent.pl` a sentence must:

- *follow* a certain character or set of characters (see Listing 300); 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 301); by default, this is only capital letters;
- *end* with a character (see Listing 302); 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 300: sentencesFollow -m

```
491         sentencesFollow:
492             par: 1
493             blankLine: 1
494             fullStop: 1
495             exclamationMark: 1
496             questionMark: 1
497             rightBrace: 1
498
      commentOnPreviousLine: 1
499             other: 0
```

LISTING 301: sentencesBeginWith -m

```
500         sentencesBeginWith:
501             A-Z: 1
502             a-z: 0
503             other: 0
```

LISTING 302: sentencesEndWith -m

```
504         sentencesEndWith:
505             basicFullStop: 0
506             betterFullStop: 1
507             exclamationMark: 1
508             questionMark: 1
509             other: 0
```

6.2.1 sentencesFollow

Let's explore a few of the switches in `sentencesFollow`; let's start with Listing 295 on the preceding page, and use the YAML settings given in Listing 304. Using the command

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

we obtain the output given in Listing 303.

LISTING 303: multiple-sentences.tex using Listing 304

```
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 304: 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 300 with the `.tex` file detailed in Listing 305.

LISTING 305: 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 306 and 307.

LISTING 306: multiple-sentences1.tex using Listing 297 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 307: multiple-sentences1.tex using Listing 308

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

LISTING 308: sentences-follow2.yaml -m

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

Notice that in Listing 306 the first sentence after the) has not been accounted for, but that following the inclusion of Listing 308, the output given in Listing 307 demonstrates that the sentence *has* been accounted for correctly.

6.2.2 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 301), and we can use the `other` field to define sentences to begin with other characters.

LISTING 309: 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 310 and 311.

LISTING 310: multiple-sentences2.tex using Listing 297 on page 84

```
This is the first sentence.

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

LISTING 311: multiple-sentences2.tex using Listing 312

```
This is the first sentence.

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

LISTING 312: sentences-begin1.yaml -m

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

Notice that in Listing 310, the first sentence has been accounted for but that the subsequent sentences have not. In Listing 311, all of the sentences have been accounted for, because the `other` field in Listing 312 has defined sentences to begin with either `$` or any numeric digit, `0` to `9`.

6.2.3 sentencesEndWith

Let's return to Listing 295 on page 83; we have already seen the default way in which `latexindent.pl` will operate on the sentences in this file in Listing 296 on page 84. We can populate the `other` field with any character that we wish; for example, using the YAML specified in Listing 314 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 313.

LISTING 313: multiple-sentences.tex using Listing 314

```
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 314: sentences-end1.yaml -m

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

LISTING 315: multiple-sentences.tex using Listing 316

```
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 316: sentences-end2.yaml -m

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

There is a subtle difference between the output in Listings 313 and 315; in particular, in Listing 313 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 316, and the associated output in Listing 315 reflects this.

Referencing Listing 302 on page 84, 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 317.

LISTING 317: 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 318.

LISTING 318: url.tex using Listing 297 on page 84

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

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

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

N: 2019-07-13

If you find that the `betterFullStop` does not work for your purposes, then you can switch it off by setting it to `0`, and you can experiment with the `other` field. You can also seek to customise the `betterFullStop` routine by using the *fine tuning*, detailed in Listing 497 on page 124.

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

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

and the YAML in Listing 320 gives the output in Listing 319.

LISTING 319: url.tex using Listing 320

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

LISTING 320: alt-full-stop1.yaml -m

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

Notice that the full stop within the URL has not been accommodated correctly because of the non-default settings in Listing 320.

6.2.4 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 321, and run the command

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

then we obtain the output in Listing 322.

LISTING 321: 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 322: multiple-sentences3.tex using Listing 297 on page 84

```
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 323 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 324 and 325.

LISTING 323: multiple-sentences4.tex

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

LISTING 324: multiple-sentences4.tex using Listing 297 on page 84

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

LISTING 325: multiple-sentences4.tex using Listing 299 on page 84

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

Once you've read Section 6.5, you will know that you can accommodate the removal of internal sentence line breaks by using the YAML in Listing 327 and the command

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

the output of which is shown in Listing 326.

LISTING 326: multiple-sentences4.tex using Listing 327

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

LISTING 327: item-rules2.yaml -m

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

6.2.5 text wrapping and indenting sentences

N: 2018-08-13

The oneSentencePerLine can be instructed to perform text wrapping and indentation upon sentences.

Let's use the code in Listing 328.

LISTING 328: multiple-sentences5.tex

```
A distinção entre conteúdo \emph{real} e conteúdo \emph{intencional} está
relacionada, ainda, à distinção entre o conceito husserliano de
\emph{experiência} e o uso popular desse termo. No sentido comum,
o \term{experimentado} é um complexo de eventos exteriores,
e o \term{experimentar} consiste em percepções (além de julgamentos e outros
atos) nas quais tais eventos aparecem como objetos, e objetos frequentemente
relacionados ao ego empírico.
```

Referencing Listing 330, and running the following command

```
cmh:~$ latexindent.pl multiple-sentences5 -m -l=sentence-wrap1.yaml
```

we receive the output given in Listing 329.

LISTING 329: multiple-sentences5.tex using Listing 330

```
A distinção entre conteúdo \emph{real} e conteúdo
  \emph{intencional} está relacionada, ainda, à
  distinção entre o conceito husserliano de
  \emph{experiência} e o uso popular desse termo.
No sentido comum, o \term{experimentado} é um
  complexo de eventos exteriores, e o
  \term{experimentar} consiste em percepções (além
  de julgamentos e outros atos) nas quais tais
  eventos aparecem como objetos, e objetos
  frequentemente relacionados ao ego empírico.
```

LISTING 330: sentence-wrap1.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        removeSentenceLineBreaks: 1
        textWrapSentences: 1
        sentenceIndent: "  "
    textWrapOptions:
        columns: 50
```

If you wish to specify the `columns` field on a per-code-block basis for sentences, then you would use `sentence`; explicitly, starting with Listing 289 on page 81, for example, you would replace/append `environments` with, for example, `sentence:  50`.

If you specify `textWrapSentences` as 1, but do *not* specify a value for `columns` then the text wrapping will *not* operate on sentences, and you will see a warning in `indent.log`.

The indentation of sentences requires that sentences are stored as code blocks. This means that you may need to tweak Listing 302 on page 84. Let's explore this in relation to Listing 331.

LISTING 331: multiple-sentences6.tex

```
Consider the following:
\begin{itemize}
        \item firstly.
        \item secondly.
\end{itemize}
```

By default, `latexindent.pl` will find the full-stop within the first `item`, which means that, upon running the following commands

```
cmh:~$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml
cmh:~$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml
    -y="modifyLineBreaks:oneSentencePerLine:sentenceIndent:'  '"
```

we receive the respective output in Listing 332 and Listing 333.

LISTING 332: multiple-sentences6-mod1.tex using Listing 330

```
Consider the following: \begin{itemize} \item
  firstly.
\item secondly.
\end{itemize}
```

LISTING 333: multiple-sentences6-mod2.tex using Listing 330 and no sentence indentation

```
Consider the following: \begin{itemize} \item
          firstly.
    \item secondly.
\end{itemize}
```

We note that Listing 332 the `itemize` code block has *not* been indented appropriately. This is because the oneSentencePerLine has been instructed to store sentences (because Listing 330); each sentence is then searched for code blocks.

We can tweak the settings in Listing 302 on page 84 to ensure that full stops are not followed by `item` commands, and that the end of sentences contains `\end{itemize}` as in Listing 334 (if you intend to

use this, ensure that you remove the line breaks from the `other` field).

LISTING 334: itemize.yaml -m

```
modifyLineBreaks:
    oneSentencePerLine:
        manipulateSentences: 1
        sentencesEndWith:
            betterFullStop: 0
            other: '(?:\.\)(?!\h*[a-z]))|(?:(?<!(?:(?:e\.g)
                    |(?:i\.e)|(?:etc))))\.(?:\h*\R*(?:\\end\{itemize\})?)
                    (?!(?:[a-z]|[A-Z]|\-|\,|[0-9]|(?:(?:\R|\h)*\\item)))'
```

Upon running

```
cmh:~$ latexindent.pl multiple-sentences6 -m -l=sentence-wrap1.yaml,itemize.yaml
```

we receive the output in Listing 335.

LISTING 335: multiple-sentences6-mod3.tex using Listing 330 and Listing 334

```
Consider the following: \begin{itemize} \item
            firstly. \item secondly.
  \end{itemize}
```

Notice that the sentence has received indentation, and that the `itemize` code block has been found and indented correctly.

6.3 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 336. Thank you to [19] for shaping and assisting with the testing of this feature.

`removeParagraphLineBreaks`: ⟨*fields*⟩

This feature is considered complimentary to the `oneSentencePerLine` feature described in Section 6.2 on page 82.

LISTING 336: removeParagraphLineBreaks -m

```
528     removeParagraphLineBreaks:
529         all: 0
530         beforeTextWrap: 0
531         alignAtAmpersandTakesPriority: 1
532         environments:
533             quotation: 0
534         ifElseFi: 0
535         optionalArguments: 0
536         mandatoryArguments: 0
537         items: 0
538         specialBeginEnd: 0
539         afterHeading: 0
540         preamble: 0
541         filecontents: 0
542         masterDocument: 0
```

This routine can be turned on *globally* for *every* code block type known to `latexindent.pl` (see Table 1 on page 49) 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 1 on page 49;
- 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 353 on page 96.

Let's start with the `.tex` file in Listing 337, together with the YAML settings in Listing 338.

LISTING 337: shortlines.tex

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

Another
paragraph.
\end{myenv}
```

LISTING 338: 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 339.

LISTING 339: 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 337 which has crept in to the output in Listing 339. This can be fixed using the YAML file in Listing 413 on page 104 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 340; notice that the double spaces present in Listing 339 have been addressed.

LISTING 340: 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 338, we note that the `all` switch applies to *all* code block types. So, for example, let's consider the files in Listings 341 and 342

LISTING 341: shortlines-mand.tex

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

Another
paragraph.
}
```

LISTING 342: 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 343 and 344.

LISTING 343: shortlines-mand1.tex

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

    Another  paragraph.
}
```

LISTING 344: 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 345, and consider the settings in Listings 346 and 347; note that in Listing 346 we specify that *every* environment should receive treatment from the routine, while in Listing 347 we specify that *only* the `one` environment should receive the treatment.

LISTING 345: 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 346: remove-para2.yaml -m

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

LISTING 347: 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 348 and 349.

LISTING 348: 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 349: 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 customised 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 350, with the YAML settings in Listing 351.

LISTING 350: 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 351: 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 352.

LISTING 352: 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}
```

U: 2018-08-13

Note that the `all` field can take the same exceptions detailed in Listing 283lst:textwrap8-yaml.

`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 353.

LISTING 353: paragraphsStopAt -m

```
543         paragraphsStopAt:
544             environments: 1
545             verbatim: 1
546             commands: 0
547             ifElseFi: 0
548             items: 0
549             specialBeginEnd: 0
550             heading: 0
551             filecontents: 0
552             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 353. By default, you'll see that the paragraph line break routine will stop when it reaches an environment or verbatim code block 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 354; we will, in turn, consider the settings in Listings 355 and 356.

LISTING 354: 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 355: stop-command.yaml -m

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

LISTING 356: stop-comment.yaml -m

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

Upon using the settings from Listing 351 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 357 to 359; notice in particular that:

- in Listing 357 the paragraph line break routine has included commands and comments;
- in Listing 358 the paragraph line break routine has *stopped* at the `emph` command, because in Listing 355 we have specified `commands` to be `1`, and `emph` is at the beginning of a line;
- in Listing 359 the paragraph line break routine has *stopped* at the comments, because in Listing 356 we have specified `comments` to be `1`, and the comment is at the beginning of a line.

In all outputs in Listings 357 to 359 we notice that the paragraph line break routine has stopped at `\begin{myenv}` because, by default, `environments` is set to `1` in Listing 353 on the preceding page.

LISTING 357: 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 358: 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 359: 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
```

6.4 Combining removeParagraphLineBreaks and textWrapOptions

N: 2018-08-13

The text wrapping routine (Section 6.1 on page 73) and remove paragraph line breaks routine (Section 6.3 on page 91) can be combined.

We motivate this feature with the code given in Listing 360.

LISTING 360: textwrap7.tex

```
This paragraph
has line breaks throughout its paragraph;
we would like to combine
the textwrapping
and paragraph removal routine.
```

Applying the text wrap routine from Section 6.1 on page 73 with, for example, Listing 277 on page 77 gives the output in Listing 361.

LISTING 361: `textwrap7.tex` using Listing 277

```
This paragraph
has line breaks throughout
its paragraph;
we would like to combine
the textwrapping
and paragraph removal
routine.
```

The text wrapping routine has behaved as expected, but it may be desired to remove paragraph line breaks *before* performing the text wrapping routine. The desired behaviour can be achieved by employing the `beforeTextWrap` switch.

Explicitly, using the settings in Listing 363 and running the command

```
cmh:~$ latexindent.pl -m textwrap7.tex -l=textwrap12.yaml -o=+-mod12
```

we obtain the output in Listing 362.

LISTING 362: `textwrap7-mod12.tex`

```
This paragraph has line
breaks throughout its
paragraph; we would like to
combine the textwrapping and
paragraph removal routine.
```

LISTING 363: `textwrap12.yaml` `-m`

```
modifyLineBreaks:
    textWrapOptions:
        columns: 30
        perCodeBlockBasis: 1
        all: 1
    removeParagraphLineBreaks:
        all: 1
        beforeTextWrap: 1
```

In Listing 362 the paragraph line breaks have first been removed from Listing 360, and then the text wrapping routine has been applied. It is envisaged that variants of Listing 363 will be among the most useful settings for these two features.

6.5 Poly-switches

U: 2017-08-21

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

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

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

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

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

N: 2017-08-21

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

N: 2019-07-13

4 *add blank line mode*; a blank line will be added before or after the *<part of thing>* under consideration, even if the *<part of thing>* is already on its own line.

In the above, *<part of thing>* refers to either the *begin statement*, *body* or *end statement* of the code blocks detailed in Table 1 on page 49. 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.

6.6 modifyLineBreaks for environments

We start by viewing a snippet of `defaultSettings.yaml` in Listing 364; note that it contains *global* settings (immediately after the `environments` field) and that *per-name* settings are also allowed – in the case of Listing 364, settings for `equation*` have been specified for demonstration. Note that all poly-switches are *off* (set to 0) by default.

LISTING 364: environments -m

```
553     environments:
554         BeginStartsOnOwnLine: 0
555         BodyStartsOnOwnLine: 0
556         EndStartsOnOwnLine: 0
557         EndFinishesWithLineBreak: 0
558         equation*:
559             BeginStartsOnOwnLine: 0
560             BodyStartsOnOwnLine: 0
561             EndStartsOnOwnLine: 0
562             EndFinishesWithLineBreak: 0
```

Let's begin with the simple example given in Listing 365; note that we have annotated key parts of the file using ♠, ♡, ♢ and ♣, these will be related to fields specified in Listing 364.

LISTING 365: env-mlb1.tex

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

6.6.1 Adding line breaks: BeginStartsOnOwnLine and BodyStartsOnOwnLine

Let's explore `BeginStartsOnOwnLine` and `BodyStartsOnOwnLine` in Listings 366 and 367, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 366: env-mlb1.yaml -m

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

LISTING 367: 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 368 and 369 respectively.

LISTING 368: env-mlb.tex using Listing 366

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

LISTING 369: env-mlb.tex using Listing 367

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

There are a couple of points to note:

- in Listing 368 a line break has been added at the point denoted by ♠ in Listing 365; no other line breaks have been changed;
- in Listing 369 a line break has been added at the point denoted by ♡ in Listing 365; furthermore, note that the *body* of `myenv` has received the appropriate (default) indentation.

Let's now change each of the 1 values in Listings 366 and 367 so that they are 2 and save them into `env-mlb3.yaml` and `env-mlb4.yaml` respectively (see Listings 370 and 371).

LISTING 370: env-mlb3.yaml -m

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

LISTING 371: env-mlb4.yaml -m

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

Upon running commands analogous to the above, we obtain Listings 372 and 373.

LISTING 372: env-mlb.tex using Listing 370

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

LISTING 373: env-mlb.tex using Listing 371

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

Note that line breaks have been added as in Listings 368 and 369, 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 366 and 367 so that they are 3 and save them into env-mlb5.yaml and env-mlb6.yaml respectively (see Listings 374 and 375).

LISTING 374: env-mlb5.yaml -m

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

LISTING 375: env-mlb6.yaml -m

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

Upon running commands analogous to the above, we obtain Listings 376 and 377.

LISTING 376: env-mlb.tex using Listing 374

```
before words

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

LISTING 377: env-mlb.tex using Listing 375

```
before words \begin{myenv}

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

Note that line breaks have been added as in Listings 368 and 369, but this time a *blank line* has been added after adding the line break.

N: 2019-07-13

Let's now change each of the 1 values in Listings 374 and 375 so that they are 4 and save them into env-beg4.yaml and env-body4.yaml respectively (see Listings 378 and 379).

LISTING 378: env-beg4.yaml -m

```
modifyLineBreaks:
    environments:
        BeginStartsOnOwnLine: 4
```

LISTING 379: env-body4.yaml -m

```
modifyLineBreaks:
    environments:
        BodyStartsOnOwnLine: 4
```

We will demonstrate this poly-switch value using the code in Listing 380.

LISTING 380: env-mlb1.tex

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

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb1.tex -l env-beg4.yaml
cmh:~$ latexindent.pl -m env-mlb.1tex -l env-body4.yaml
```

then we receive the respective outputs in Listings 381 and 382.

LISTING 381: env-mlb1.tex using Listing 378

```
before words

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

LISTING 382: env-mlb1.tex using Listing 379

```
before words
\begin{myenv}

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

We note in particular that, by design, for this value of the poly-switches:

1. in Listing 381 a blank line has been inserted before the `\begin` statement, even though the `\begin` statement was already on its own line;
2. in Listing 382 a blank line has been inserted before the beginning of the *body*, even though it already began on its own line.

6.6.2 Adding line breaks using EndStartsOnOwnLine and EndFinishesWithLineBreak

Let's explore `EndStartsOnOwnLine` and `EndFinishesWithLineBreak` in Listings 383 and 384, and in particular, let's allow each of them in turn to take a value of 1.

LISTING 383: env-mlb7.yaml -m

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

LISTING 384: 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 385 and 386.

LISTING 385: env-mlb.tex using Listing 383

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

LISTING 386: env-mlb.tex using Listing 384

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

There are a couple of points to note:

- in Listing 385 a line break has been added at the point denoted by ♢ in Listing 365 on page 99; no other line breaks have been changed and the `\end{myenv}` statement has *not* received indentation (as intended);
- in Listing 386 a line break has been added at the point denoted by ♣ in Listing 365 on page 99.

Let's now change each of the 1 values in Listings 383 and 384 so that they are 2 and save them into `env-mlb9.yaml` and `env-mlb10.yaml` respectively (see Listings 387 and 388).

LISTING 387: env-mlb9.yaml -m

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

LISTING 388: env-mlb10.yaml -m

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

Upon running commands analogous to the above, we obtain Listings 389 and 390.

LISTING 389: env-mlb.tex using Listing 387

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

LISTING 390: env-mlb.tex using Listing 388

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

Note that line breaks have been added as in Listings 385 and 386, 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 383 and 384 so that they are 3 and save them into `env-mlb11.yaml` and `env-mlb12.yaml` respectively (see Listings 391 and 392).

LISTING 391: env-mlb11.yaml -m

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

LISTING 392: env-mlb12.yaml -m

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

Upon running commands analogous to the above, we obtain Listings 393 and 394.

LISTING 393: env-mlb.tex using Listing 391

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

\end{myenv} after words
```

LISTING 394: env-mlb.tex using Listing 392

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

after words
```

Note that line breaks have been added as in Listings 385 and 386, and that a *blank line* has been added after the line break.

N: 2019-07-13

Let's now change each of the 1 values in Listings 391 and 392 so that they are 4 and save them into env-end4.yaml and env-end-f4.yaml respectively (see Listings 395 and 396).

LISTING 395: env-end4.yaml -m

```
modifyLineBreaks:
    environments:
        EndStartsOnOwnLine: 4
```

LISTING 396: env-end-f4.yaml -m

```
modifyLineBreaks:
    environments:
        EndFinishesWithLineBreak: 4
```

We will demonstrate this poly-switch value using the code from Listing 380 on page 100.

Upon running the commands

```
cmh:~$ latexindent.pl -m env-mlb1.tex -l env-end4.yaml
cmh:~$ latexindent.pl -m env-mlb.1tex -l env-end-f4.yaml
```

then we receive the respective outputs in Listings 397 and 398.

LISTING 397: env-mlb1.tex using Listing 395

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

\end{myenv}
after words
```

LISTING 398: env-mlb1.tex using Listing 396

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

after words
```

We note in particular that, by design, for this value of the poly-switches:

1. in Listing 397 a blank line has been inserted before the \end statement, even though the \end statement was already on its own line;
2. in Listing 398 a blank line has been inserted after the \end statement, even though it already began on its own line.

6.6.3 poly-switches 1, 2, and 3 only add line breaks when necessary

If you ask latexindent.pl to add a line break (possibly with a comment) using a poly-switch value of 1 (or 2 or 3), it will only do so if necessary. For example, if you process the file in Listing 399 using poly-switch values of 1, 2, or 3, it will be left unchanged.

LISTING 399: env-mlb2.tex

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

LISTING 400: env-mlb3.tex

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

Setting the poly-switches to a value of 4 instructs latexindent.pl to add a line break even if the *<part of thing>* is already on its own line; see Listings 381 and 382 and Listings 397 and 398.

In contrast, the output from processing the file in Listing 400 will vary depending on the poly-switches used; in Listing 401 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 402 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 400 and by setting the other poly-switches considered so far to 2 in turn.

LISTING 401: env-mlb3.tex using Listing 367 on page 99

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

LISTING 402: env-mlb3.tex using Listing 371 on page 99

```
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 364 on page 99, an example is shown for the equation* environment.

6.6.4 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 403, noting in particular the positions of the line break highlighters, ♠, ♡, ♢ and ♣, together with the associated YAML files in Listings 404 to 407.

LISTING 403: env-mlb4.tex

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

LISTING 404: env-mlb13.yaml -m

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

LISTING 405: env-mlb14.yaml -m

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

LISTING 406: env-mlb15.yaml -m

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

LISTING 407: 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 408 to 411.

LISTING 408: env-mlb4.tex using Listing 404

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

LISTING 409: env-mlb4.tex using Listing 405

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

LISTING 410: env-mlb4.tex using Listing 406

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

LISTING 411: env-mlb4.tex using Listing 407

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

Notice that in:

- Listing 408 the line break denoted by ♠ in Listing 403 has been removed;
- Listing 409 the line break denoted by ♡ in Listing 403 has been removed;
- Listing 410 the line break denoted by ♢ in Listing 403 has been removed;
- Listing 411 the line break denoted by ♣ in Listing 403 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 404 to 407 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 365 on page 99.

6.6.5 About trailing horizontal space

Recall that on page 30 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 412, which highlights trailing spaces.

LISTING 412: env-mlb5.tex

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

LISTING 413: 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 414 and 415; note that the trailing horizontal white space has been preserved (by default) in Listing 414, while in Listing 415, it has been removed using the switch specified in Listing 413.

LISTING 414: env-mlb5.tex using Listings 408 to 411

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

LISTING 415: env-mlb5.tex using Listings 408 to 411 *and* Listing 413

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

6.6.6 poly-switch line break removal and blank lines

Now let's consider the file in Listing 416, which contains blank lines.

LISTING 416: env-mlb6.tex

```
before words♠


\begin{myenv}♡


body of myenv♢


\end{myenv}♣

after words
```

LISTING 417: 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 418 and 419. In Listing 418 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 419, we have allowed the poly-switches to remove blank lines because, in Listing 417, we have set `preserveBlankLines` to 0.

LISTING 418: env-mlb6.tex using Listings 408 to 411

```
before words

\begin{myenv}

    body of myenv

\end{myenv}

after words
```

LISTING 419: env-mlb6.tex using Listings 408 to 411 *and* Listing 417

```
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 420.

LISTING 420: 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 421 and 422.

LISTING 421: env-mlb7-preserve.tex

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

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

LISTING 422: env-mlb7-no-preserve.tex

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

Notice that in:

- Listing 421 that `\end{one}` has added a blank line, because of the value of `EndFinishesWithLineBreak` in Listing 392 on page 101, and even though the line break ahead of `\begin{two}` should have been removed (because of `BeginStartsOnOwnLine` in Listing 404 on page 103), the blank line has been preserved by default;
- Listing 422, by contrast, has had the additional line-break removed, because of the settings in Listing 417.

6.7 Poly-switches for double back slash

N: 2019-07-13

With reference to `lookForAlignDelims` (see Listing 33 on page 30) you can specify poly-switches to dictate the line-break behaviour of double back slashes in environments (Listing 35 on page 30), commands (Listing 69 on page 37), or special code blocks (Listing 104 on page 42). Note that for these poly-switches to take effect, the name of the code block must necessarily be specified within `lookForAlignDelims` (Listing 33 on page 30); we will demonstrate this in what follows.

Consider the code given in Listing 423.

LISTING 423: tabular3.tex

```
\begin{tabular}{cc}
 1 & 2 ★\\□ 3 & 4 ★\\□
\end{tabular}
```

Referencing Listing 423:

- DBS stands for *double back slash*;
- line breaks ahead of the double back slash are annotated by ★, and are controlled by `DBSStartsOnOwnLine`;
- line breaks after the double back slash are annotated by □, and are controlled by `DBSFinishesWithLineBreak`.

Let's explore each of these in turn.

6.7.1 Double back slash starts on own line

We explore `DBSStartsOnOwnLine` (★ in Listing 423); starting with the code in Listing 423, together with the YAML files given in Listing 425 and Listing 427 and running the following commands

```
cmh:~$ latexindent.pl -m tabular3.tex -l DBS1.yaml
cmh:~$ latexindent.pl -m tabular3.tex -l DBS2.yaml
```

then we receive the respective output given in Listing 424 and Listing 426.

LISTING 424: tabular3.tex using Listing 425

```
\begin{tabular}{cc}
    1 & 2
    \\ 3 & 4
    \\
\end{tabular}
```

LISTING 425: DBS1.yaml -m

```
modifyLineBreaks:
    environments:
        DBSStartsOnOwnLine: 1
```

LISTING 426: tabular3.tex using Listing 427

```
\begin{tabular}{cc}
    1 & 2 %
    \\ 3 & 4%
    \\
\end{tabular}
```

LISTING 427: DBS2.yaml -m

```
modifyLineBreaks:
    environments:
        tabular:
            DBSStartsOnOwnLine: 2
```

We note that

- Listing 425 specifies `DBSStartsOnOwnLine` for *every* environment (that is within `lookForAlignDelims`, Listing 36 on page 31); the double back slashes from Listing 423 have been moved to their own line in Listing 424;
- Listing 427 specifies `DBSStartsOnOwnLine` on a *per-name* basis for `tabular` (that is within `lookForAlignDelims`, Listing 36 on page 31); the double back slashes from Listing 423 have been moved to their own line in Listing 426, having added comment symbols before moving them.

6.7.2 Double back slash finishes with line break

Let's now explore `DBSFinishesWithLineBreak` (□ in Listing 423); starting with the code in Listing 423, together with the YAML files given in Listing 429 and Listing 431 and running the following commands

```
cmh:~$ latexindent.pl -m tabular3.tex -l DBS3.yaml
cmh:~$ latexindent.pl -m tabular3.tex -l DBS4.yaml
```

then we receive the respective output given in Listing 428 and Listing 430.

LISTING 428: tabular3.tex using Listing 429

```
\begin{tabular}{cc}
    1 & 2 \\
    3 & 4 \\
\end{tabular}
```

LISTING 429: DBS3.yaml -m

```
modifyLineBreaks:
    environments:
        DBSFinishesWithLineBreak: 1
```

LISTING 430: tabular3.tex using Listing 431

```
\begin{tabular}{cc}
    1 & 2 \\%
    3 & 4 \\
\end{tabular}
```

LISTING 431: DBS4.yaml -m

```
modifyLineBreaks:
    environments:
        tabular:
            DBSFinishesWithLineBreak: 2
```

We note that

- Listing 429 specifies `DBSFinishesWithLineBreak` for *every* environment (that is within `lookForAlignDelims`, Listing 36 on page 31); the code following the double back slashes from Listing 423 has been moved to their own line in Listing 428;
- Listing 431 specifies `DBSFinishesWithLineBreak` on a *per-name* basis for `tabular` (that is within `lookForAlignDelims`, Listing 36 on page 31); the first double back slashes from Listing 423 have moved code following them to their own line in Listing 430, having added comment symbols before moving them; the final double back slashes have *not* added a line break as they are at the end of the body within the code block.

6.7.3 Double back slash poly-switches for specialBeginEnd

Let's explore the double back slash poly-switches for code blocks within `specialBeginEnd` code blocks (Listing 102 on page 42); we begin with the code within Listing 432.

LISTING 432: special4.tex

```
\< a& =b  \\ & =c\\ & =d\\ & =e \>
```

Upon using the YAML settings in Listing 434, and running the command

```
cmh:~$ latexindent.pl -m special4.tex -l DBS5.yaml
```

then we receive the output given in Listing 433.

LISTING 433: special4.tex using Listing 434

```
\<
    a & =b \\
      & =c \\
      & =d \\
      & =e %
\>
```

LISTING 434: DBS5.yaml

-m

```
specialBeginEnd:
    cmhMath:
        lookForThis: 1
        begin: '\\<'
        end: '\\>'
lookForAlignDelims:
    cmhMath: 1
modifyLineBreaks:
    specialBeginEnd:
        cmhMath:
            DBSFinishesWithLineBreak: 1
            SpecialBodyStartsOnOwnLine: 1
            SpecialEndStartsOnOwnLine: 2
```

There are a few things to note:

- in Listing 434 we have specified `cmhMath` within `lookForAlignDelims`; without this, the double back slash poly-switches would be ignored for this code block;
- the `DBSFinishesWithLineBreak` poly-switch has controlled the line breaks following the double back slashes;
- the `SpecialEndStartsOnOwnLine` poly-switch has controlled the addition of a comment symbol, followed by a line break, as it is set to a value of 2.

6.7.4 Double back slash poly-switches for optional and mandatory arguments

For clarity, we provide a demonstration of controlling the double back slash poly-switches for optional and mandatory arguments. We begin with the code in Listing 435.

LISTING 435: mycommand2.tex

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

Upon using the YAML settings in Listings 437 and 439, and running the command

```
cmh:~$ latexindent.pl -m mycommand2.tex -l DBS6.yaml
cmh:~$ latexindent.pl -m mycommand2.tex -l DBS7.yaml
```

then we receive the output given in Listings 436 and 438.

LISTING 436: mycommand2.tex using Listing 437

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

LISTING 437: DBS6.yaml -m

```
lookForAlignDelims:
    mycommand: 1
modifyLineBreaks:
    optionalArguments:
        DBSStartsOnOwnLine: 2
        DBSFinishesWithLineBreak: 2
```

LISTING 438: mycommand2.tex using Listing 439

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

LISTING 439: DBS7.yaml -m

```
lookForAlignDelims:
    mycommand: 1
modifyLineBreaks:
    mandatoryArguments:
        DBSStartsOnOwnLine: 2
        DBSFinishesWithLineBreak: 2
```

6.7.5 Double back slash optional square brackets

The pattern matching for the double back slash will also, optionally, allow trailing square brackets that contain a measurement of vertical spacing, for example `\\[3pt]`.

For example, beginning with the code in Listing 440

LISTING 440: pmatrix3.tex

```
\begin{pmatrix}
1 & 2 \\[2pt] 3 & 4 \\ [ 3 ex] 5&6\\[    4  pt   ] 7 & 8
\end{pmatrix}
```

and running the following command, using Listing 429,

```
cmh:~$ latexindent.pl -m pmatrix3.tex -l DBS3.yaml
```

then we receive the output given in Listing 441.

LISTING 441: pmatrix3.tex using Listing 429

```
\begin{pmatrix}
    1 & 2 \\[2pt]
    3 & 4 \\ [ 3 ex]
    5 & 6 \\[    4  pt   ]
    7 & 8
\end{pmatrix}
```

You can customise the pattern for the double back slash by exploring the *fine tuning* field detailed in Listing 497 on page 124.

6.8 Poly-switches for other code blocks

Rather than repeat the examples shown for the environment code blocks (in Section 6.6 on page 99), we choose to detail the poly-switches for all other code blocks in Table 2; note that each and every one of these poly-switches is *off by default*, i.e, set to 0.

U: 2019-05-05

Note also that, by design, line breaks involving, `filecontents` and 'comment-marked' code blocks (Listing 70 on page 37) can *not* be modified using `latexindent.pl`. However, there are two poly-switches available for `verbatim` code blocks: environments (Listing 18 on page 27), commands (Listing 19 on page 27) and `specialBeginEnd` (Listing 115 on page 44).

TABLE 2: 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	`...`♠
	`[`♡
	`value before comma`★`,`
	□
	`end of body of opt arg`♢
	`]`♣
	`...`
		♠
		♡
		★
		□
		♢
		♣
			LSqBStartsOnOwnLine[8]
			OptArgBodyStartsOnOwnLine
			CommaStartsOnOwnLine
			CommaFinishesWithLineBreak
			RSqBStartsOnOwnLine
			RSqBFinishesWithLineBreak

mandatoryArguments	`...`♠
	`{`♡
	`value before comma`★`,`
	□
	`end of body of mand arg`♢
	`}`♣
	`...`
		♠
		♡
		★
		□
		♢
		♣
			LCuBStartsOnOwnLine[9]
			MandArgBodyStartsOnOwnLine
			CommaStartsOnOwnLine
			CommaFinishesWithLineBreak
			RCuBStartsOnOwnLine
			RCuBFinishesWithLineBreak

commands	`before words`♠ `\mycommand`♡ ⟨*arguments*⟩	♠ ♡	CommandStartsOnOwnLine CommandNameFinishesWithLineB
namedGroupingBraces Brackets	before words♠ myname♡ ⟨*braces/brackets*⟩	♠ ♡	NameStartsOnOwnLine NameFinishesWithLineBreak
keyEqualsValuesBraces Brackets	before words♠ key•=♡ ⟨*braces/brackets*⟩	♠ • ♡	KeyStartsOnOwnLine EqualsStartsOnOwnLine EqualsFinishesWithLineBreak
items	before words♠ `\item`♡ `...`	♠ ♡	ItemStartsOnOwnLine ItemFinishesWithLineBreak
specialBeginEnd	before words♠ `\[`♡ `body of special/middle`★ `\middle`□ body of special/middle ♢ `\]`♣ after words	♠ ♡ ★ □ ♢ ♣	SpecialBeginStartsOnOwnLine SpecialBodyStartsOnOwnLine SpecialMiddleStartsOnOwnLine SpecialMiddleFinishesWithLineBre SpecialEndStartsOnOwnLine SpecialEndFinishesWithLineBreak
verbatim	before words♠`\begin{verbatim}`	♠	VerbatimBeginStartsOnOwnLine

N: 2018-04-27
N: 2019-07-13
N: 2019-07-13
N: 2018-04-27

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

N: 2019-05-05

body of verbatim \end{verbatim}♣ ♣ VerbatimEndFinishesWithLineBreal
after words

6.9 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 2 on the preceding page) and `LCuBStartsOnOwnLine` for mandatory arguments, and `LSqBStartsOnOwnLine` for optional arguments.

Let's begin with the code in Listing 442 and the YAML settings in Listing 444; with reference to Table 2 on the previous page, the key `CommandNameFinishesWithLineBreak` is an alias for `BodyStartsOnOwnLine`.

LISTING 442: 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 443; note that the *second* mandatory argument beginning brace { has had its leading line break removed, but that the *first* brace has not.

LISTING 443: mycommand1.tex using Listing 444

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

LISTING 444: 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 446; upon running the analogous command to that given above, we obtain Listing 445; both beginning braces { have had their leading line breaks removed.

LISTING 445: mycommand1.tex using Listing 446

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

LISTING 446: 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 448; upon running the analogous command to that given above, we obtain Listing 447.

LISTING 447: mycommand1.tex using Listing 448

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

LISTING 448: mycom-mlb3.yaml -m

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

6.10 Conflicting poly-switches: sequential code blocks

It is very easy to have conflicting poly-switches; if we use the example from Listing 442 on the preceding page, and consider the YAML settings given in Listing 450. The output from running

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

is given in Listing 450.

LISTING 449: mycommand1.tex using Listing 450

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

LISTING 450: mycom-mlb4.yaml -m

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

Studying Listing 450, 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 449, 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 452; upon running the command

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

we obtain the output given in Listing 451.

LISTING 451: mycommand1.tex using Listing 452

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

LISTING 452: 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 454, which give associated output in Listing 453.

LISTING 453: mycommand1.tex using Listing 454

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

LISTING 454: 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.

6.11 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 455, noting that it contains nested environments.

LISTING 455: nested-env.tex

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

Let's use the YAML settings given in Listing 457, which upon running the command

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

gives the output in Listing 456.

LISTING 456: nested-env.tex using Listing 457

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

LISTING 457: nested-env-mlb1.yaml -m

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

In Listing 456, 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 455, 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 456, 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 459; upon running the command

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

we obtain the output given in Listing 458.

LISTING 458: `nested-env.tex` using Listing 459

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

LISTING 459: `nested-env-mlb2.yaml` -m

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

During Phase 1:

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

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

SECTION 7

The -r, -rv and -rr switches

N: 2019-07-13

You can instruct `latexindent.pl` to perform replacements/substitutions on your file by using any of the `-r`, `-rv` or `-rr` switches:

- the `-r` switch will perform indentation and replacements, not respecting verbatim code blocks;
- the `-rv` switch will perform indentation and replacements, and *will* respect verbatim code blocks;
- the `-rr` switch will *not* perform indentation, and will perform replacements not respecting verbatim code blocks.

We will demonstrate each of the `-r`, `-rv` and `-rr` switches, but a summary is given in Table 3.

TABLE 3: The replacement mode switches

switch	indentation?	respect verbatim?
`-r`	✔	✖
`-rv`	✔	✔
`-rr`	✖	✖

The default value of the `replacements` field is shown in Listing 460; as with all of the other fields, you are encouraged to customise and change this as you see fit. The options in this field will *only* be considered if the `-r`, `-rv` or `-rr` switches are active; when discussing YAML settings related to the replacement-mode switches, we will use the style given in Listing 460.

LISTING 460: `replacements` -r

```
614 replacements:
615   -
616     amalgamate: 1
617   -
618     this: 'latexindent.pl'
619     that: 'pl.latexindent'
620     lookForThis: 1
621     when: before
```

The first entry within the `replacements` field is `amalgamate`, and is *optional*; by default it is set to 1, so that replacements will be amalgamated from each settings file that you specify. As you'll see in the demonstrations that follow, there is no need to specify this field.

You'll notice that, by default, there is only *one* entry in the `replacements` field, but it can take as many entries as you would like; each one needs to begin with a `-` on its own line.

7.1 Introduction to replacements

Let's explore the action of the default settings, and then we'll demonstrate the feature with further examples. With reference to Listing 460, the default action will replace every instance of the text `latexindent.pl` with `pl.latexindent`.

Beginning with the code in Listing 461 and running the command

```
cmh:~$ latexindent.pl -r replace1.tex
```

gives the output given in Listing 462.

LISTING 461: replace1.tex

```
Before text, latexindent.pl,
after text.
```

LISTING 462: replace1.tex default

```
Before text, pl.latexindent,
after text.
```

If we don't wish to perform this replacement, then we can tweak the default settings of Listing 460 on the previous page by changing `lookForThis` to 0; we perform this action in Listing 464, and run the command

```
cmh:~$ latexindent.pl -r replace1.tex -l=replace1.yaml
```

which gives the output in Listing 463.

LISTING 463: replace1.tex using Listing 464

```
Before text, latexindent.pl,
after text.
```

LISTING 464: replace1.yaml -r

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

Note that in Listing 464 we have specified `amalgamate` as 0 so that the default replacements are overwritten.

We haven't yet discussed the `when` field; don't worry, we'll get to it as part of the discussion in what follows.

7.2 The two types of replacements

There are two types of replacements:

1. *string*-based replacements, which replace the string in *this* with the string in *that*. If you specify `this` and you do not specify `that`, then the `that` field will be assumed to be empty.
2. *regex*-based replacements, which use the `substitution` field.

We will demonstrate both in the examples that follow.

`latexindent.pl` chooses which type of replacement to make based on which fields have been specified; if the `this` field is specified, then it will make *string*-based replacements, regardless of if `substitution` is present or not.

7.3 Examples of replacements

Example 1 We begin with code given in Listing 465

LISTING 465: colsep.tex

```
\begin{env}
1 2 3\arraycolsep=3pt
4 5 6\arraycolsep=5pt
\end{env}
```

Let's assume that our goal is to remove both of the `arraycolsep` statements; we can achieve this in a few different ways.

Using the YAML in Listing 467, and running the command

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

then we achieve the output in Listing 466.

LISTING 466: colsep.tex using Listing 465

```
\begin{env}
    1 2 3
    4 5 6
\end{env}
```

LISTING 467: colsep.yaml -r

```
replacements:
  -
    this: \arraycolsep=3pt
  -
    this: \arraycolsep=5pt
```

Note that in Listing 467, we have specified *two* separate fields, each with their own '*this*' field; furthermore, for both of the separate fields, we have not specified 'that', so the that field is assumed to be blank by latexindent.pl;

We can make the YAML in Listing 467 more concise by exploring the substitution field. Using the settings in Listing 469 and running the command

```
cmh:~$ latexindent.pl -r colsep.tex -l=colsep1.yaml
```

then we achieve the output in Listing 468.

LISTING 468: colsep.tex using Listing 469

```
\begin{env}
    1 2 3
    4 5 6
\end{env}
```

LISTING 469: colsep1.yaml -r

```
replacements:
  -
    substitution: s/\\arraycolsep=\d+pt//sg
```

The code given in Listing 469 is an example of a *regular expression*, which we may abbreviate to *regex* in what follows. This manual is not intended to be a tutorial on regular expressions; you might like to read, for example, [10] for a detailed covering of the topic. With reference to Listing 469, we do note the following:

- the general form of the substitution field is s/regex/replacement/modifiers. You can place any regular expression you like within this;
- we have 'escaped' the backslash by using \\
- we have used \d+ to represent *at least* one digit
- the s *modifier* (in the sg at the end of the line) instructs latexindent.pl to treat your file as one single line;
- the g *modifier* (in the sg at the end of the line) instructs latexindent.pl to make the substitution *globally* throughout your file; you might try removing the g modifier from Listing 469 and observing the difference in output.

You might like to see https://perldoc.perl.org/perlre.html#Modifiers for details of modifiers; in general, I recommend starting with the sg modifiers for this feature.

Example 2 We'll keep working with the file in Listing 465 on the preceding page for this example.

Using the YAML in Listing 471, and running the command

```
cmh:~$ latexindent.pl -r colsep.tex -l=multi-line.yaml
```

then we achieve the output in Listing 470.

LISTING 470: colsep.tex using Listing 471

```
multi-line!
```

LISTING 471: multi-line.yaml -r

```
replacements:
  -
    this: |-
     \begin{env}
     1 2 3\arraycolsep=3pt
     4 5 6\arraycolsep=5pt
     \end{env}
    that: 'multi-line!'
```

With reference to Listing 471, we have specified a *multi-line* version of `this` by employing the *literal* YAML style `|-`. See, for example, https://stackoverflow.com/questions/3790454/in-yaml-how-do-i-break-a-string-over-multiple-lines for further options, all of which can be used in your YAML file.

This is a natural point to explore the `when` field, specified in Listing 460 on page 115. This field can take two values: *before* and *after*, which respectively instruct `latexindent.pl` to perform the replacements *before* indentation or *after* it. The default value is `before`.

Using the YAML in Listing 473, and running the command

```
cmh:~$ latexindent.pl -r colsep.tex -l=multi-line1.yaml
```

then we achieve the output in Listing 472.

LISTING 472: colsep.tex using Listing 473

```
\begin{env}
    1 2 3\arraycolsep=3pt
    4 5 6\arraycolsep=5pt
\end{env}
```

LISTING 473: multi-line1.yaml -r

```
replacements:
  -
    this: |-
     \begin{env}
     1 2 3\arraycolsep=3pt
     4 5 6\arraycolsep=5pt
     \end{env}
    that: 'multi-line!'
    when: after
```

We note that, because we have specified `when: after`, that `latexindent.pl` has not found the string specified in Listing 473 within the file in Listing 465 on page 116. As it has looked for the string within Listing 473 *after* the indentation has been performed. After indentation, the string as written in Listing 473 is no longer part of the file, and has therefore not been replaced.

As a final note on this example, if you use the `-rr` switch, as follows,

```
cmh:~$ latexindent.pl -rr colsep.tex -l=multi-line1.yaml
```

then the `when` field is ignored, no indentation is done, and the output is as in Listing 470.

Example 3 An important part of the substitution routine is in *capture groups*.

Assuming that we start with the code in Listing 474, let's assume that our goal is to replace each occurrence of `$$...$$` with `\begin{equation*}...\end{equation*}`. This example is partly motivated by tex stackexchange question 242150.

LISTING 474: displaymath.tex

```
before text $$a^2+b^2=4$$ and $$c^2$$

$$
d^2+e^2 = f^2
$$
and also $$ g^2
$$ and some inline math: $h^2$
```

We use the settings in Listing 476 and run the command

```
cmh:~$ latexindent.pl -r displaymath.tex -l=displaymath1.yaml
```

to receive the output given in Listing 475.

LISTING 475: displaymath.tex using Listing 476

```
before text \begin{equation*}a^2+b^2=4\end{equation*}

\begin{equation*}
    d^2+e^2 = f^2
\end{equation*}
and also \begin{equation*} g^2
\end{equation*} and some inline math: $h^2$
```

LISTING 476: displaymath1.yaml

-r

```
replacements:
  -
    substitution: |-
     s/\$\$
       (.*?)
       \$\$/\\begin{equation*}$1\\end{equation*}/sgx
```

A few notes about Listing 476:

1. we have used the `x` modifier, which allows us to have white space within the regex;
2. we have used a capture group, `(.*?)` which captures the content between the `$$...$$` into the special variable, `$1`;
3. we have used the content of the capture group, `$1`, in the replacement text.

See https://perldoc.perl.org/perlre.html#Capture-groups for a discussion of capture groups.

The features of the replacement switches can, of course, be combined with others from the toolkit of `latexindent.pl`. For example, we can combine the poly-switches of Section 6.5 on page 98, which we do in Listing 478; upon running the command

```
cmh:~$ latexindent.pl -r -m displaymath.tex -l=displaymath1.yaml,equation.yaml
```

then we receive the output in Listing 477.

LISTING 477: displaymath.tex using Listings 476 and 478

```
before text%
\begin{equation*}%
    a^2+b^2=4%
\end{equation*}%
and%
\begin{equation*}%
    c^2%
\end{equation*}

\begin{equation*}
    d^2+e^2 = f^2
\end{equation*}
and also%
\begin{equation*}%
    g^2
\end{equation*}%
and some inline math: $h^2$
```

LISTING 478: equation.yaml -m

```
modifyLineBreaks:
    environments:
        equation*:
            BeginStartsOnOwnLine: 2
            BodyStartsOnOwnLine: 2
            EndStartsOnOwnLine: 2
            EndFinishesWithLineBreak: 2
```

Example 4 This example is motivated by tex stackexchange question 490086. We begin with the code in Listing 479.

LISTING 479: phrase.tex

```
phrase 1       phrase 2 phrase 3            phrase 100

phrase 1       phrase 2 phrase 3            phrase 100

phrase 1       phrase 2 phrase 3            phrase 100

phrase 1       phrase 2 phrase 3            phrase 100
```

Our goal is to make the spacing uniform between the phrases. To achieve this, we employ the settings in Listing 481, and run the command

```
cmh:~$ latexindent.pl -r phrase.tex -l=hspace.yaml
```

which gives the output in Listing 480.

LISTING 480: phrase.tex using Listing 481

```
phrase 1 phrase 2 phrase 3 phrase 100

phrase 1 phrase 2 phrase 3 phrase 100

phrase 1 phrase 2 phrase 3 phrase 100

phrase 1 phrase 2 phrase 3 phrase 100
```

LISTING 481: hspace.yaml -r

```
replacements:
  -
    substitution: s/\h+/ /sg
```

The \h+ setting in Listing 481 say to replace *at least one horizontal space* with a single space.

Example 5 We begin with the code in Listing 482.

LISTING 482: references.tex

```
equation \eqref{eq:aa} and Figure  \ref{fig:bb}
and table~\ref{tab:cc}
```

Our goal is to change each reference so that both the text and the reference are contained within one hyperlink. We achieve this by employing Listing 484 and running the command

```
cmh:~$ latexindent.pl -r references.tex -l=reference.yaml
```

which gives the output in Listing 483.

LISTING 483: references.tex using Listing 484

```
\hyperref{equation \ref*{eq:aa}} and \hyperref{Figure \ref*{fig:bb}}
and \hyperref{table \ref*{tab:cc}}
```

LISTING 484: reference.yaml -r

```
replacements:
  -
    substitution: |-
      s/(
        equation
        |
        table
        |
        figure
        |
        section
      )
      (\h|~)*
      \\(?:eq)?
      ref\{(.*?)\}/\\hyperref{$1 \\ref\*{$3}}/sgxi
```

Referencing Listing 484, the `|` means *or*, we have used *capture groups*, together with an example of an *optional* pattern, `(?:eq)?`.

Example 6 Let's explore the three replacement mode switches (see Table 3 on page 115) in the context of an example that contains a verbatim code block, Listing 485; we will use the settings in Listing 486.

LISTING 485: verb1.tex

```
\begin{myenv}
body of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    body
        of
      verbatim
 text
\end{verbatim}
text
```

LISTING 486: verbatim1.yaml -r

```
replacements:
  -
    this: 'body'
    that: 'head'
```

Upon running the following commands,

```
cmh:~$ latexindent.pl -r verb1.tex -l=verbatim1.yaml -o=+mod1
cmh:~$ latexindent.pl -rv verb1.tex -l=verbatim1.yaml -o=+-rv-mod1
cmh:~$ latexindent.pl -rr verb1.tex -l=verbatim1.yaml -o=+-rr-mod1
```

we receive the respective output in Listings 487 to 489

LISTING 487: verb1-mod1.tex

```
\begin{myenv}
    head of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    head
        of
      verbatim
 text
\end{verbatim}
text
```

LISTING 488: verb1-rv-mod1.tex

```
\begin{myenv}
    head of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    body
        of
      verbatim
 text
\end{verbatim}
text
```

LISTING 489: verb1-rr-mod1.tex

```
\begin{myenv}
head of verbatim
\end{myenv}
some verbatim
\begin{verbatim}
    head
        of
      verbatim
 text
\end{verbatim}
text
```

We note that:

1. in Listing 487 indentation has been performed, and that the replacements specified in Listing 486 have been performed, even within the verbatim code block;
2. in Listing 488 indentation has been performed, but that the replacements have *not* been performed within the verbatim environment, because the `rv` switch is active;
3. in Listing 489 indentation has *not* been performed, but that replacements have been performed, not respecting the verbatim code block.

See the summary within Table 3 on page 115.

Example 7 Let's explore the `amalgamate` field from Listing 460 on page 115 in the context of the file specified in Listing 490.

LISTING 490: amalg1.tex

```
one two three
```

Let's consider the YAML files given in Listings 491 to 493.

LISTING 491: amalg1-yaml.yaml -r

```
replacements:
  -
    this: one
    that: 1
```

LISTING 492: amalg2-yaml.yaml -r

```
replacements:
  -
    this: two
    that: 2
```

LISTING 493: amalg3-yaml.yaml -r

```
replacements:
  -
    amalgamate: 0
  -
    this: three
    that: 3
```

Upon running the following commands,

```
cmh:~$ latexindent.pl -r amalg1.tex -l=amalg1-yaml
cmh:~$ latexindent.pl -r amalg1.tex -l=amalg1-yaml,amalg2-yaml
cmh:~$ latexindent.pl -r amalg1.tex -l=amalg1-yaml,amalg2-yaml,amalg3-yaml
```

we receive the respective output in Listings 494 to 496.

LISTING 494: `amalg1.tex` using Listing 491

```
1 two three
```

LISTING 495: `amalg1.tex` using Listings 491 and 492

```
1 2 three
```

LISTING 496: `amalg1.tex` using Listings 491 to 493

```
one two 3
```

We note that:

1. in Listing 494 the replacements from Listing 491 have been used;
2. in Listing 495 the replacements from Listings 491 and 492 have *both* been used, because the default value of `amalgamate` is 1;
3. in Listing 496 *only* the replacements from Listing 493 have been used, because the value of `amalgamate` has been set to 0.

SECTION 8

Fine tuning

N: 2019-07-13

`latexindent.pl` operates by looking for the code blocks detailed in Table 1 on page 49. The fine tuning of the details of such code blocks is controlled by the `fineTuning` field, detailed in Listing 497.

This field is for those that would like to peek under the bonnet/hood and make some fine tuning to `latexindent.pl`'s operating.

Making changes to the fine tuning may have significant consequences for your indentation scheme, proceed with caution!

LISTING 497: `fineTuning`

```
625 fineTuning:
626     environments:
627       name: '[a-zA-Z@\*0-9_\\]+'
628     ifElseFi:
629       name: '(?!@?if[a-zA-Z@]*?\{)@?if[a-zA-Z@]*?'
630     commands:
631       name: '[+a-zA-Z@\*0-9_\:]+?'
632     keyEqualsValuesBracesBrackets:
633       name: '[a-zA-Z@\*0-9_\/.:\#-]+[a-zA-Z@\*0-9_\/.\h\{\}:\#-]*?'
634       follow: '(?:(?<!\\)\{)|,|(?:(?<!\\)\[)'
635     namedGroupingBracesBrackets:
636       name: '[0-9\.a-zA-Z@\*><]+?'
637       follow: '\h|\R|\{|\[|\$|\)|\('
638     UnNamedGroupingBracesBrackets:
639       follow: '\{|\[|,|&|\)|\(|\$'
640     arguments:
641       before: '(?:#\d\h*;?,?\/?)+|\<.*?\>'
642       between: '_|\^|\*'
643     trailingComments:
644       notPreceededBy: '(?<!\\)'
645     modifyLineBreaks:
646       betterFullStop:
      '(?:\.\)(?!\h*[a-z]))|(?:(?<!(?:(?:e\.g)|(?:i\.e)|(?:etc))))\.(?!(?:[a-z]|[A-Z]|\-|~|\,|[0-9]))'
647       doubleBackSlash: '\\\\(?:\h*\[\h*\d+\h*[a-zA-Z]+\h*\])?'
648       comma: ','
```

The fields given in Listing 497 are all *regular expressions*. This manual is not intended to be a tutorial on regular expressions; you might like to read, for example, [10] for a detailed covering of the topic.

We make the following comments with reference to Listing 497:

1. the `environments:name` field details that the *name* of an environment can contain:
   (a) `a-z` lower case letters
   (b) `A-Z` upper case letters
   (c) `@` the `@` 'letter'
   (d) `\*` stars
   (e) `0-9` numbers

(f) _ underscores

(g) \ backslashes

The + at the end means *at least one* of the above characters.

2. the `ifElseFi:name` field:
   (a) `@?` means that it *can possibly* begin with `@`
   (b) followed by `if`
   (c) followed by 0 or more characters from `a-z`, `A-Z` and `@`
   (d) the `?` the end means *non-greedy*, which means 'stop the match as soon as possible'
3. the `keyEqualsValuesBracesBrackets` contains some interesting syntax:
   (a) `|` means 'or'
   (b) `(?:(?<!\\)\{)` the `(?:...)` uses a *non-capturing* group – you don't necessarily need to worry about what this means, but just know that for the `fineTuning` feature you should only ever use *non*-capturing groups, and *not* capturing groups, which are simply `(...)`
   (c) `(?<!\\)\{)` means a `{` but it can *not* be immediately preceded by a `\`
4. in the `arguments:before` field
   (a) `\d\h*` means a digit (i.e. a number), followed by 0 or more horizontal spaces
   (b) `;?,?` means *possibly* a semi-colon, and possibly a comma
   (c) `\<.*?\>` is designed for 'beamer'-type commands; the `.*?` means anything in between `<...>`
5. the `modifyLineBreaks` field refers to fine tuning settings detailed in Section 6 on page 72. In particular:
   (a) `betterFullStop` is in relation to the one sentence per line routine, detailed in Section 6.2 on page 82
   (b) `doubleBackSlash` is in relation to the `DBSStartsOnOwnLine` and `DBSFinishesWithLineBreak` polyswitches surrounding double back slashes, see Section 6.7 on page 106
   (c) `comma` is in relation to the `CommaStartsOnOwnLine` and `CommaFinishesWithLineBreak` polyswitches surrounding commas in optional and mandatory arguments; see Table 2 on page 110

It is not obvious from Listing 497, but each of the `follow`, `before` and `between` fields allow trailing comments, line breaks, and horizontal spaces between each character.

Example 8 As a demonstration, consider the file given in Listing 498, together with its default output using the command

```
cmh:~$ latexindent.pl finetuning1.tex
```

is given in Listing 499.

LISTING 498: `finetuning1.tex`

```
\mycommand{
    \rule{G -> +H[-G]CL}
    \rule{H -> -G[+H]CL}
    \rule{g -> +h[-g]cL}
    \rule{h -> -g[+h]cL}
}
```

LISTING 499: `finetuning1.tex` default

```
\mycommand{
\rule{G -> +H[-G]CL}
\rule{H -> -G[+H]CL}
\rule{g -> +h[-g]cL}
\rule{h -> -g[+h]cL}
}
```

It's clear from Listing 499 that the indentation scheme has not worked as expected. We can *fine tune* the indentation scheme by employing the settings given in Listing 501 and running the command

```
cmh:~$ latexindent.pl finetuning1.tex -l=fine-tuning1.yaml
```

and the associated (desired) output is given in Listing 500.

LISTING 500: finetuning1.tex using Listing 501

```
\mycommand{
    \rule{G -> +H[-G]CL}
    \rule{H -> -G[+H]CL}
    \rule{g -> +h[-g]cL}
    \rule{h -> -g[+h]cL}
}
```

LISTING 501: finetuning1.yaml

```
fineTuning:
    arguments:
      between:
    '_|\^|\*|\->|\-|\+|h|H|g|G'
```

Example 9 Let's have another demonstration; consider the file given in Listing 502, together with its default output using the command

```
cmh:~$ latexindent.pl finetuning2.tex
```

is given in Listing 503.

LISTING 502: finetuning2.tex

```
@misc{ wikilatex,
author = "{Wikipedia contributors}",
title = "LaTeX --- {Wikipedia}{,}",
note = "[Online; accessed 3-March-2020]"
}
```

LISTING 503: finetuning2.tex default

```
@misc{ wikilatex,
author = "{Wikipedia contributors}",
title = "LaTeX --- {Wikipedia}{,}",
note = "[Online; accessed 3-March-2020]"
}
```

It's clear from Listing 503 that the indentation scheme has not worked as expected. We can *fine tune* the indentation scheme by employing the settings given in Listing 505 and running the command

```
cmh:~$ latexindent.pl finetuning2.tex -l=fine-tuning2.yaml
```

and the associated (desired) output is given in Listing 504.

LISTING 504: finetuning2.tex using Listing 505

```
@misc{ wikilatex,
    author = "{Wikipedia contributors}",
    title = "LaTeX --- {Wikipedia}{,}",
    note = "[Online; accessed 3-March-2020]"
}
```

LISTING 505: finetuning2.yaml

```
fineTuning:
    NamedGroupingBracesBrackets:
      follow: '\h|\R|\{|\[|\$|\)|\(|"'
    UnNamedGroupingBracesBrackets:
      follow: '\{|\[|,|&|\)|\(|\$|"'
    arguments:
      between: '_|\^|\*|---'
```

In particular, note that the settings in Listing 505 specify that `NamedGroupingBracesBrackets` and `UnNamedGroupingBracesBrackets` can follow " and that we allow --- between arguments.

Example 10 You can tweak the `fineTuning` using the `-y` switch, but to be sure to use quotes appropriately. For example, starting with the code in Listing 506 and running the following command

```
cmh:~$ latexindent.pl -m
    -y='modifyLineBreaks:oneSentencePerLine:manipulateSentences:␣1,␣
    modifyLineBreaks:oneSentencePerLine:sentencesBeginWith:a-z:␣1,␣
    fineTuning:modifyLineBreaks:betterFullStop:␣
    "(?:\.|;|:(?![a-z]))|(?:(?<!(?:(?:e\.g)|(?:i\.e)|(?:etc))))\.(?!(?:[a-z]|[A-Z]|\~|
    issue-243.tex -o=+-mod1
```

gives the output shown in Listing 507.

LISTING 506: `finetuning3.tex`

```
We go; you see: this sentence \cite{tex:stackexchange} finishes here.
```

LISTING 507: `finetuning3.tex` using -y switch

```
We go;
you see:
this sentence \cite{tex:stackexchange} finishes here.
```

Example 11 We can tweak the `fineTuning` for how trailing comments are classified. For motivation, let's consider the code given in Listing 508

LISTING 508: `finetuning4.tex`

```
some before text
 \href{Handbook%20for%30Spoken%40document.pdf}{my document}
some after text
```

We will compare the settings given in Listings 509 and 510.

LISTING 509: `href1.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 80
        huge: overflow
        all: 1
        perCodeBlockBasis: 1
    removeParagraphLineBreaks:
        all: 1
```

LISTING 510: `href2.yaml` -m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 80
        huge: overflow
        all: 1
        perCodeBlockBasis: 1
    removeParagraphLineBreaks:
        all: 1

fineTuning:
    trailingComments:
      notPreceededBy:
    '(?:(?<!Handbook)(?<!for)(?<!Spoken))'
```

Upon running the following commands

```
cmh:~$ latexindent.pl -m finetuning4.tex -o=+-mod1 -l=href1
cmh:~$ latexindent.pl -m finetuning4.tex -o=+-mod2 -l=href2
```

we receive the respective output in Listings 511 and 512.

LISTING 511: `finetuning4.tex` using Listing 509

```
some before text \href{Handbook some after text%20for%30Spoken%40document.pdf}{my document}
```

LISTING 512: finetuning4.tex using Listing 510

```
some before text \href{Handbook%20for%30Spoken%40document.pdf}{my document} some after text
```

We note that in:

- Listing 511 the trailing comments are assumed to be everything following the first comment symbol, which has meant that everything following it has been moved to the end of the line; this is undesirable, clearly!
- Listing 512 has fine-tuned the trailing comment matching, and says that % cannot be immediately preceeded by the words 'Handbook', 'for' or 'Spoken', which means that none of the % symbols have been treated as trailing comments, and the output is desirable.

Another approach to this situation, which does not use `fineTuning`, is to use `noIndentBlock` which we discussed in Listing 20 on page 27; using the settings in Listing 513 and running the command

```
cmh:~$ latexindent.pl -m finetuning4.tex -o=+-mod3 -l=href3
```

then we receive the same output given in Listing 512; see also `paragraphsStopAt` in Listing 353 on page 96.

LISTING 513: href3.yaml

-m

```
modifyLineBreaks:
    textWrapOptions:
        columns: 80
        huge: overflow
        all: 1
        perCodeBlockBasis: 1
    removeParagraphLineBreaks:
        all: 1
        paragraphsStopAt:
            verbatim: 0

noIndentBlock:
    href:
        begin: '\\href\{[^}]*?\}\{'
        body: '[^}]*?'
        end: '\}'
```

With reference to the `body` field in Listing 513, we note that `[^}]*?` can be interpreted as: the fewest number of zero or more characters that are not right braces. This is an example of character class.

SECTION 9

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 in page 31; for example, consider the file given in Listing 514.

LISTING 514: `matrix2.tex`

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

The default output is given in Listing 515, and it is clear that the alignment routine has not worked as hoped, but it is *expected*.

LISTING 515: `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 1 on page 49) 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 46 on page 33); in particular, when working with code blocks 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 page 113); 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.

U: 2019-07-13

You can run `latexindent` on any file; if you don't specify an extension, then the extensions that you specify in `fileExtensionPreference` (see Listing 16 on page 25) will be consulted. If you find a case in which the script struggles, please feel free to report it at [11], and in the meantime, consider using a `noIndentBlock` (see page 27).

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 [11]; otherwise, feel free to find me on the http://tex.stackexchange.com/users/6621/cmhughes.

Section 10

References

10.1 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).

[5] *CPAN: Comprehensive Perl Archive Network*. URL: http://www.cpan.org/ (visited on 01/23/2017).

[6] *Data Dumper demonstration*. URL: https://stackoverflow.com/questions/7466825/how-do-you-sort-the-output-of-datadumper (visited on 06/18/2021).

[7] *Data::Dumper module*. URL: https://perldoc.perl.org/Data::Dumper (visited on 06/18/2021).

[10] Jeffrey E. F. Friedl. *Mastering Regular Expressions*. ISBN: 0596002890.

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

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

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

[21] *perldoc Encode::Supported*. URL: https://perldoc.perl.org/Encode::Supported (visited on 05/06/2021).

[24] *Strawberry Perl*. URL: http://strawberryperl.com/ (visited on 01/23/2017).

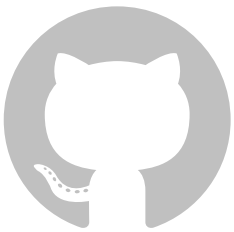
[25] *Text::Tabs Perl module*. URL: http://search.cpan.org/~muir/Text-Tabs+Wrap-2013.0523/lib.old/Text/Tabs.pm (visited on 07/06/2017).

[26] *Text::Wrap Perl module*. URL: http://perldoc.perl.org/Text/Wrap.html (visited on 05/01/2017).

[27] *Video demonstration of latexindent.pl on youtube*. URL: https://www.youtube.com/watch?v=wo38aaH2F4E&spfreload=10 (visited on 02/21/2017).

[29] *Windows line breaks on Linux prevent removal of white space from end of line*. URL: https://github.com/cmhughes/latexindent.pl/issues/256 (visited on 06/18/2021).

10.2 Contributors



[2] Sam Abey. *Print usage tip to STDERR only if STDIN is TTY*. Sept. 17, 2019. URL: https://github.com/cmhughes/latexindent.pl/pull/174 (visited on 03/19/2021).

[3] Paulo Cereda. *arara rule, indent.yaml*. May 23, 2013. URL: https://github.com/islandoftex/arara/blob/master/rules/arara-rule-indent.yaml (visited on 03/19/2021).

[4] Cheng Xu (xu cheng). *always output log/help text to STDERR*. July 13, 2018. URL: https://github.com/cmhughes/latexindent.pl/pull/121 (visited on 08/05/2018).

[8] 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).

[9] Jacobo Diaz. *Hiddenconfig*. July 21, 2014. URL: https://github.com/cmhughes/latexindent.pl/pull/18 (visited on 01/23/2017).

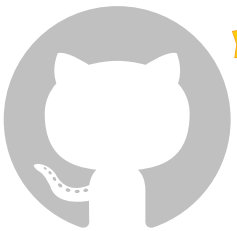
[12] Randolf J. *Alpine-linux instructions*. Aug. 10, 2020. URL: https://github.com/cmhughes/latexindent.pl/pull/214 (visited on 08/10/2020).

[13] jeanlego. *Search localSettings in CWD as well*. Sept. 15, 2020. URL: https://github.com/cmhughes/latexindent.pl/pull/221 (visited on 03/19/2021).

[14] Jason Juang. *add in PATH installation*. Nov. 24, 2015. URL: https://github.com/cmhughes/latexindent.pl/pull/38 (visited on 01/23/2017).

[15] Harish Kumar. *Early version testing*. Nov. 10, 2013. URL: https://github.com/harishkumarholla (visited on 06/30/2017).

[17] mlep. *One sentence per line*. Aug. 16, 2017. URL: https://github.com/cmhughes/latexindent.pl/issues/81 (visited on 01/08/2018).



[18] newptcai. *Update appendices.tex*. Feb. 16, 2021. URL: https://github.com/cmhughes/latexindent.pl/pull/221 (visited on 03/19/2021).

[19] John Owens. *Paragraph line break routine removal*. May 27, 2017. URL: https://github.com/cmhughes/latexindent.pl/issues/33 (visited on 05/27/2017).

[22] qiancy98. *Locale encoding of file system*. May 6, 2021. URL: https://github.com/cmhughes/latexindent.pl/pull/273 (visited on 05/06/2021).

[23] Alexander Regueiro. *Update help screen*. Mar. 18, 2021. URL: https://github.com/cmhughes/latexindent.pl/pull/261 (visited on 03/19/2021).

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

[30] Tom Zöhner (zoehneto). *Improving text wrap*. Feb. 4, 2018. URL: https://github.com/cmhughes/latexindent.pl/issues/103 (visited on 08/04/2018).

SECTION 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 516 (`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 516: `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 Encode;
use Getopt::Long;
use Data::Dumper;
use List::Util qw(max);

print "hello world";
exit;
```

A.1 Module installer script

N: 2018-01-13

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

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

or

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

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

A.2 Manually installed modules

Manually installing the modules given in Listing 516 will vary depending on your operating system and `Perl` distribution.

A.2.1 Linux

Linux users may be interested in exploring Perlbrew [20]; an example installation would be:

```
cmh:~$ sudo apt-get install perlbrew
cmh:~$ perlbrew init
cmh:~$ perlbrew install perl-5.28.1
cmh:~$ perlbrew switch perl-5.28.1
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
```

For other distributions, the Ubuntu/Debian approach may work as follows

```
cmh:~$ sudo apt install perl
cmh:~$ sudo cpan -i App::cpanminus
cmh:~$ sudo cpanm YAML::Tiny
cmh:~$ sudo cpanm File::HomeDir
cmh:~$ sudo cpanm Unicode::GCString
```

or else by running, for example,

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

If you are using Alpine, some `Perl` modules are not build-compatible with Alpine, but replacements are available through `apk`. For example, you might use the commands given in Listing 517; thanks to [12] for providing these details.

LISTING 517: `alpine-install.sh`

```
# Installing perl
apk --no-cache add miniperl perl-utils

# Installing incompatible latexindent perl dependencies via apk
apk --no-cache add \
    perl-log-dispatch \
    perl-namespace-autoclean \
    perl-specio \
    perl-unicode-linebreak

# Installing remaining latexindent perl dependencies via cpan
apk --no-cache add curl wget make
ls /usr/share/texmf-dist/scripts/latexindent
cd /usr/local/bin && \
    curl -L https://cpanmin.us/ -o cpanm && \
    chmod +x cpanm
cpanm -n App::cpanminus
cpanm -n File::HomeDir
cpanm -n Params::ValidationCompiler
cpanm -n YAML::Tiny
cpanm -n Unicode::GCString
```

Users of NixOS might like to see https://github.com/cmhughes/latexindent.pl/issues/222 for tips.

A.2.2 Mac

Users of the Macintosh operating system might like to explore the following commands, for example:

```
cmh:~$ brew install perl
cmh:~$ brew install cpanm
cmh:~$
cmh:~$ cpanm YAML::Tiny
cmh:~$ cpanm File::HomeDir
cmh:~$ cpanm Unicode::GCString
```

A.2.3 Windows

Strawberry Perl users on Windows might use `CPAN client`. All of the modules are readily available on CPAN [5].

`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
```

SECTION B

Updating the path variable

`latexindent.pl` has a few scripts (available at [11]) that can update the `path` variables. Thank you to [14] for this feature. If you're on a Linux or Mac machine, then you'll want `CMakeLists.txt` from [11].

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 [11] ;
2. within your directory, create a directory called `path-helper-files` and download `CMakeLists.txt` and `cmake_uninstall.cmake.in` from [11]/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 [11] to your chosen directory;
2. open a command prompt and run the following command to see what is *currently* in your `%path%` variable;

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

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

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

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

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

SECTION C

logFilePreferences

Listing 17 on page 26 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 518, and the settings specified in Listing 519.

LISTING 518: simple.tex

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

LISTING 519: 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 520.

LISTING 520: indent.log

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

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

SECTION D

Encoding indentconfig.yaml

In relation to Section 4 on page 21, Windows users that encounter encoding issues with `indentconfig.yaml`, may wish to run the following command in either `cmd.exe` or `powershell.exe`:

```
C:\Users\cmh>chcp
```

They may receive the following result

```
C:\Users\cmh>Active code page: 936
```

and can then use the settings given in Listing 521 within their `indentconfig.yaml`, where 936 is the result of the `chcp` command.

LISTING 521: `encoding` demonstration for `indentconfig.yaml`

```
encoding: cp936
```

Section E

dos2unix linebreak adjustment

`dos2unixlinebreaks`: ⟨*integer*⟩

N: 2021-06-19

If you use `latexindent.pl` on a dos-based Windows file on Linux then you may find that trailing horizontal space is not removed as you hope.

In such a case, you may wish to try setting `dos2unixlinebreaks` to 1 and employing, for example, the following command.

```
cmh:~$ latexindent.pl -y="dos2unixlinebreaks:1" myfile.tex
```

See [29] for further dertails.

SECTION F

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 522 are *obsolete* from Version 3.0 onwards.

LISTING 522: 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 523 and 524

LISTING 523: `indentAfterThisHeading` in Version 2.2

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

LISTING 524: `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 525; as of Version 3.0, you would write YAML as in Listing 526 or, if you're using `-m` switch, Listing 527.

LISTING 525: noAdditionalIndent in Version 2.2

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

LISTING 526: noAdditionalIndent for displayMath in Version 3.0

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

LISTING 527: noAdditionalIndent for displayMath in Version 3.0

```
noAdditionalIndent:
    displayMath: 1
```

End

Index

— **B** —
backup files
cycle through, 26
extension settings, 25
maximum number of backup files, 26
number of backup files, 25, 26
overwrite switch, -w, 14

— **C** —
capturing parenthesis (regex), 39

— **D** —
delimiters, 107
advanced settings, 30
advanced settings of lookForAlignDelims, 30
ampersand &, 30
default settings of lookForAlignDelims, 30
delimiter justification (left or right), 39
delimiterRegEx, 39
dontMeasure feature, 37
double backslash demonstration, 36
lookForAlignDelims, 30
poly-switches for double back slash, 106
spacing demonstration, 32
with specialBeginEnd and the -m switch, 108
within specialBeginEnd blocks, 45

— **I** —
indentation
customising indentation per-code block, 48
customising per-name, 48
default, 17
defaultIndent description, 29
defaultIndent using -y switch, 17
defaultIndent using YAML file, 21
maximum indetation, 47
no additional indent, 48
no additional indent global, 48
removing indentation per-code block, 48
summary, 66

— **L** —
linebreaks
summary of poly-switches, 107

— **M** —
modifying linebreaks
at the *beginning* of a code block, 99
at the *end* of a code block, 101
by text wrapping, globally, 73
by text wrapping, per-code-block, 77
by using one sentence per line, 82
surrounding double back slash, 106
using poly-switches, 98

— **P** —
poly-switches
adding blank lines (again!): set to 4, 100, 102
adding blank lines: set to 3, 100, 101
adding comments and then line breaks: set to 2, 99, 101
adding line breaks: set to 1, 99, 101
blank lines, 105
conflicting partnering, 111
conflicting switches, 112, 113
default values, 99
definition, 98
double backslash, 107
environment global example, 99
environment per-code block example, 99
for double back slash (delimiters), 106–109
off by default: set to 0, 98
removing line breaks: set to -1, 103
summary of all poly-switches, 109
values, 98
visualisation: ♠, ♡, ♢, ♣, 99

— **R** —
regular expressions
a word about, 12
ampersand alignment, 30
arguments, 124
at least one +, 45, 117, 120, 124–126
capturing parenthesis, 39
character class demonstration, 128
commands, 124
delimiter alignment for edge or node, 45
delimiter regex at #, 40
delimiter regex at # or \>, 41
delimiter regex at \= or \>, 39
delimiter regex at only \>, 40
delimiterRegEx, 30
dontMeasure feature, cell, 38
dontMeasure feature, row, 39
environments, 124
fine tuning, 124
horizontal space \h, 45, 48, 91, 120, 121, 124
ifElseFi, 124
keyEqualsValuesBracesBrackets, 124
lowercase alph a-z, 38, 39, 48, 82, 84, 86, 91, 124

modifyLineBreaks, 124
NamedGroupingBracesBrackets, 124
numeric 0-9, 45, 48, 86, 91, 124
replacement switch, -r, 116
substitution field, arraycolsep, 117
substitution field, equation, 119
UnNamedGroupingBracesBrackets, 124
uppercase alph A-Z, 45, 48, 82, 84, 91, 124
using -y switch, 23

— **S** —
sentences
begin with, 84, 85
end with, 84, 86
follow, 84
indenting, 89
one sentence per line, 82
oneSentencePerLine, 82
removing sentence line breaks, 83
text wrapping, 89
specialBeginEnd
alignment at delimiter, 45
combined with lookForAlignDelims, 45
default settings, 42
delimiterRegEx, 45
delimiterRegEx tweaked, 45
double backslash poly-switch demonstration, 107
IfElsFi example, 43
indentRules example, 61
indentRulesGlobal, 66
introduction, 42
lookForAlignDelims, 107
middle, 43
noAdditionalIndent, 61
noAdditionalIndentGlobal, 66
paragraphsStopAt, 96
poly-switch summary, 109
removeParagraphLineBreaks, 91
searching for special before commands, 42
specifying as verbatim, 44
textWrapOptions, 77
tikz example, 45
update to displaymath V3.0, 140
switches
-c, –cruft definition and details, 18
-d, –onlydefault definition and details, 18
-g, –logfile definition and details, 18
-h, –help definition and details, 14
-l demonstration, 22, 23, 32, 37–41, 43–47, 50–63, 67–70, 74–76, 78, 79, 81, 83–96, 98–109, 111–114, 116–122, 126
-l in relation to other settings, 23
-l, –local definition and details, 16
-m demonstration, 73–76, 78, 79, 81, 83–96, 98–109, 111–114, 119
-m, –modifylinebreaks definition and details, 18
-o demonstration, 36, 40, 41, 45, 74–76, 92–96, 98, 121, 140
-o, –output definition and details, 15
-r demonstration, 115–122
-r, –replacement definition and details, 19
-rr demonstration, 118, 121
-rr, –onlyreplacement definition and details, 19
-rv demonstration, 121
-rv, –replacementrespectverb definition and details, 19
-s, –silent definition and details, 16
-sl, –screenlog definition and details, 18
-t, –trace definition and details, 16
-tt, –ttrace definition and details, 16
-v, –version definition and details, 14
-w, –overwrite definition and details, 14
-y demonstration, 23, 36, 90
-y, –yaml definition and details, 17

— **V** —
verbatim
commands, 27
comparison with -r and -rr switches, 121
environments, 27
in relation to oneSentencePerLine, 88
in relation to paragraphsStopAt, 96
in relation to textWrapOptions, 74
noIndentBlock, 27
poly-switch summary, 109
rv, replacementrespectverb switch, 19, 115
specialBeginEnd, 44
verbatimEnvironments demonstration (-l switch), 23
verbatimEnvironments demonstration (-y switch), 23
within summary of text wrapping, 82

— **W** —
warning
amalgamate field, 71
be sure to test before use, 10
capturing parenthesis for lookForAlignDelims, 39
editing YAML files, 22
fine tuning, 124
the m switch, 73