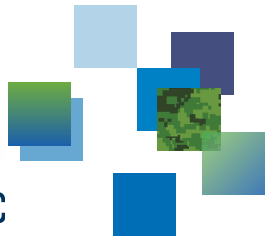
Defence Research and Development Canada
Recherche et développement pour la défense Canada



DRDC | RDDC
technologysciencetechnologie

DeLaTeXify: Grammar-Checking Enabler for LaTeX

User Manual

S. Guillouzic
DRDC – Centre for Operational Research and Analysis

Defence Research and Development Canada
Reference Document
DRDC-RDDC-2021-D076
October 2021

Canada

IMPORTANT INFORMATIVE STATEMENTS

This document was reviewed for Controlled Goods by Defence Research and Development Canada (DRDC) using the Schedule to the *Defence Production Act*.

Endorsement statement: This publication has been published by the Editorial Office of Defence Research and Development Canada, an agency of the Department of National Defence of Canada. Inquiries can be sent to: Publications.DRDC-RDDC@drdc-rddc.gc.ca.

Abstract

DeLaTeXify is a Python-based software that converts LaTeX documents to plain text, so that their grammar can be verified using tools such as Microsoft Word. It is based on text replacement rules defined using regular expressions. Users can define additional rules to process commands that are not yet supported by DeLaTeXify or are defined locally. They can also override the rules provided with DeLaTeXify if desired. The user manual provides installation and usage instructions, as well as information about how to define and debug new text replacement rules.

Résumé

DeLaTeXify est un logiciel programmé en langage Python qui convertit les documents LaTeX en texte brut, de façon à ce que l'on puisse vérifier la grammaire avec des outils comme Microsoft Word. Il est fondé sur des règles de remplacement de texte définies à l'aide d'expressions régulières. Les utilisateurs peuvent définir des règles additionnelles pour traiter les commandes qui ne sont pas encore prises en charge par DeLaTeXify ou qui sont définies localement. Ils peuvent aussi remplacer les règles fournies avec DeLaTeXify au besoin. Le manuel de l'utilisateur contient les instructions d'installation et d'utilisation, ainsi que de l'information sur la façon de définir et de déboguer de nouvelles règles de remplacement de texte.

Table of contents

List of figures

Acknowledgements

The author wishes to thank Fred Ma and Joshua Goldman for their help in beta testing DeLaTeXify.

1 Introduction

While many text editors used with LaTeX can perform spellchecking, support for grammar checking is very limited. DeLaTeXify aims to fill that gap. It converts LaTeX files to plain text in such a way as to minimize the number of false positives when checking grammar with Microsoft Word. Alternate grammar checking software can also be used on the resulting text file. Rather than try to mimic the output from LaTeX, as done by tools such as TeX4ht [1] or Pandoc [2], DeLaTeXify extracts the text that can be checked for grammar and discards the rest. For instance, each equation is replaced by an empty pair of dollar signs ($$). This notation does not interfere with the grammar checking done by Word and is easily recognized by LaTeX users as representing equations. Similarly, cross-references obtained using **\ref**, **\eqref** and **\cite** are respectively replaced with generic X, (X) and [X], since the exact numbers are irrelevant for checking grammar.

An example of a conversion performed with DeLaTeXify is shown in Figure 1, with the LaTeX source and the DeLaTeXify output respectively shown in Figures 1a and 1b. Each paragraph is wrapped by DeLaTeXify into a single line, with paragraphs separated by empty lines. As indicated above, the equation and the cross-reference are respectively replaced with

```
\documentclass{article}

\usepackage{amsmath}

\begin{document}
The first sentence introduces an equation:
\begin{equation}
  1 + 1 = 2.
  \label{eq:trivial}
\end{equation}
The second sentence ends the paragraph\footnote{What a short paragraph!}
and refers to equation~\eqref{eq:trivial}.

The second paragraph is even shorter and only has one sentence.
\end{document}
```

(a) *Source document.*

```
The first sentence introduces an equation: $$. The second sentence
ends the paragraph and refers to equation (X). (What a short
paragraph!)

The second paragraph is even shorter and only has one sentence.
```

(b) *Converted document.*

Figure 1: *Example of LaTeX-to-text conversion with DeLaTeXify.*

$$ and (X). In addition to this, the footnote is moved to the end of the paragraph and put in parentheses to avoid interrupting sentences and triggering false grammar errors. Finally, the `\documentclass`, `\usepackage`, `\begin{document}` and `\end{document}` commands are removed.

The conversion is performed by applying a set of replacement rules based on regular expressions [3]. Regular expressions provide a formal syntax to define sophisticated search patterns that can be used to extract information from a text or to identify parts where text substitutions should be made. In DeLaTeXify, such replacement rules have been defined for the more common LaTeX commands. More rules can be added as required in order to support additional commands.

Sections 2 and 3 of this manual respectively indicate how to install and run DeLaTeXify. They are the only two sections required reading in order to start using the tool. Section 4 then describes how to customize DeLaTeXify by creating new rules to process more LaTeX commands, if required. Finally, Section 5 provides concluding remarks. Annexes A to C cover more advanced subjects that are not required for basic usage: the definition of a local repository of rules in Annex A, the order of rule application in Annex B and syntax errors in rule definitions in Annex C.

2 Installation

This section describes how to install DeLaTeXify. Section 2.1 summarizes the software requirements, and Section 2.2 lists the files included in the distribution. Sections 2.3 and 2.4 go over the steps required to install it on Windows and Linux, respectively.

2.1 Software requirements

DeLaTeXify runs on both Python 2 and 3. It was tested with Python 3 on Windows 7 and with Python 2 and 3 on Linux. It processes regular expressions using the third-party *regex* module [4] if available, but defaults back to the standard *re* module [5] if not. While the *regex* module allows DeLaTeXify to run about twice as fast, the *re* module is present in all Python distributions and thus ensures portability. As graphical user interface, DeLaTeXify uses Tkinter [6], which is also a standard component of all Python distributions.

2.2 Installation files

DeLaTeXify is distributed as a set of five files, of which only the first one is absolutely required:

- **delatexify.py:** The Python script itself;
- **delatexify.ico:** Icon used for DeLaTeXify on Windows (, which is the letter L for LaTeX inscribed in the letter G for grammar);
- **Quick Start.txt:** Concise installation and usage instructions;
- **Change Log.txt:** List of changes between versions; and
- **User Manual.pdf:** This user manual.

2.3 Microsoft Windows

To install DeLaTeXify on Windows:

1. Save delatexify.py and delatexify.ico to a destination folder of your choice;
2. Run delatexify.py without any command-line argument in order to launch its Graphical User Interface (GUI);
3. In the main window of the DeLaTeXify GUI, click on the *Shortcuts* button to spawn the shortcuts dialog (see Section 3.1.1);
4. In the shortcuts dialog, choose where application shortcuts should be created and click *Create*; and

5. Drag one of the shortcuts to the taskbar to pin it there.

Steps 2 to 5 are optional, but creating shortcuts streamlines usage by allowing drag-and-drop. If the delatexify.py and delatexify.ico files are moved or if the version of Python used to run the application is changed, Steps 2 to 5 must be repeated.

If DeLaTeXify is to be used in Command-Line Interface (CLI) mode on Windows, its installation directory should be added to the PYTHONPATH environment variable and the Python installation directory should be listed in the PATH environment variable.

2.4 Linux

To install DeLaTeXify on Linux, save delatexify.py to a directory that is listed in the PATH environment variable and make sure that the file has execute permission.

3 Usage

DeLaTeXify can be used in GUI or CLI mode, respectively described in Sections 3.1 and 3.2. Section 3.3 lists some elements to keep in mind when using DeLaTeXify.

3.1 Graphical User Interface (GUI)

Screenshots of the DeLaTeXify GUI are shown in Figure 2: Windows in Figure 2a and Linux in Figure 2b. As can be seen from these screenshots, the main difference between the two interfaces is the presence of three extra buttons in Windows (*Check*, *Email log* and *Shortcuts*) for capabilities that have been implemented only for that operating system.

The GUI is split into three sections: *File names*, *Debugging options* and *Debugging log*. They are described in more detail in the following three subsections.

3.1.1 File names

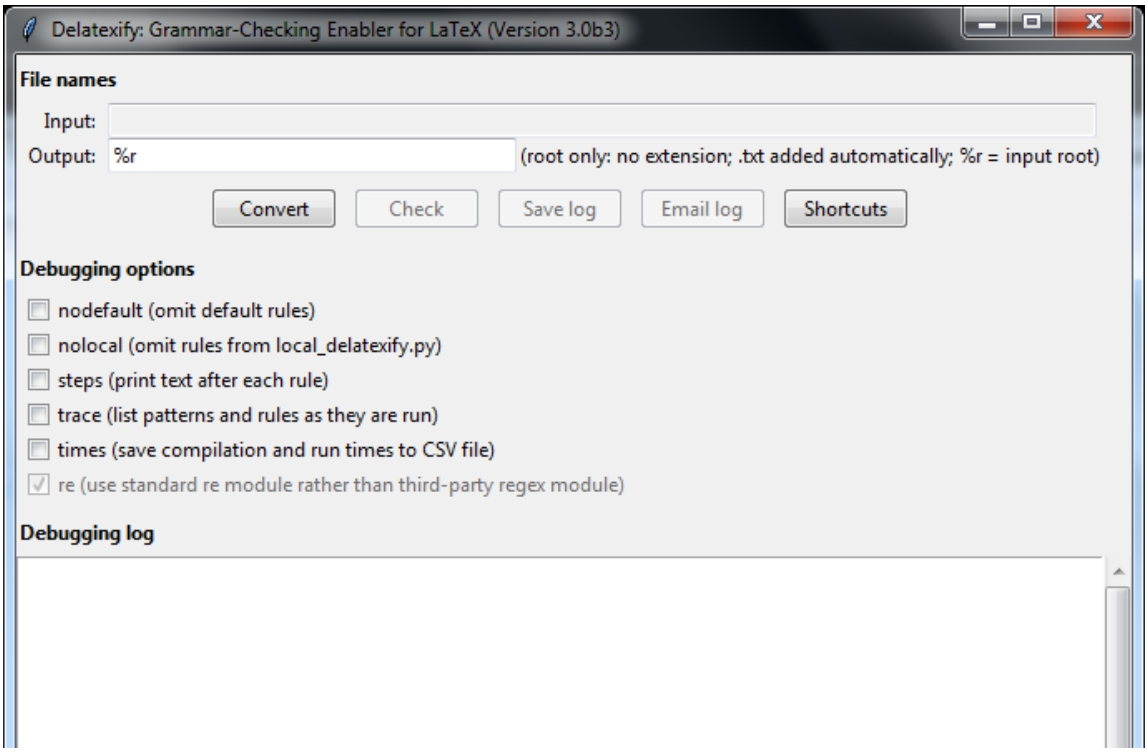
The *File names* section is where the input and output file names are specified and where buttons for running DeLaTeXify are located. Clicking on the input field spawns a dialog box where the user can choose the input file. It is populated automatically when a LaTeX file is dragged-and-dropped on one of the icons generated during installation. The output field contains the pattern used to generate the name of the output file. In the pattern, %r stands for the root name of the input file (in other words, the file name without extension). The *.txt* is added automatically to the output file name and must not be included in the output field. The default value of the output field is %r, which means that an input file name of *myreport.tex* would result in an output file name of *myreport.txt*.

The first button is called *Convert*. Clicking it launches the conversion process, with diagnostic messages written to the *Debugging log* section. The debugging log can be saved by clicking the *Save log* button. On Windows, the user has two more options once the conversion is done:

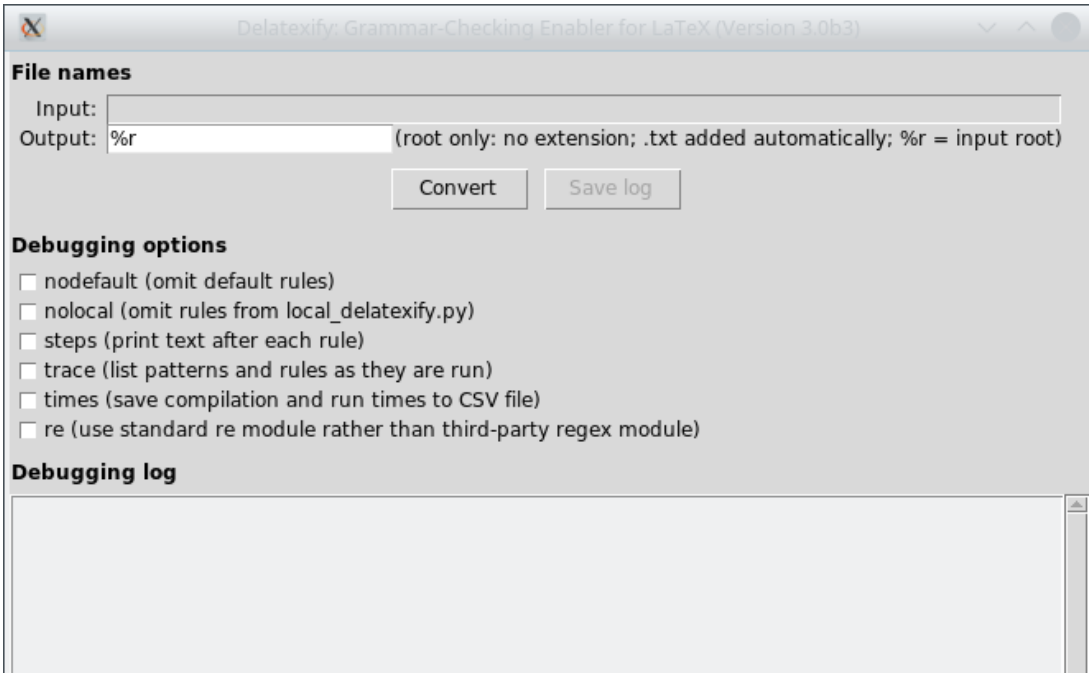
1. Click the *Check* button to open the output file in Word and initiate the grammar checking; or
2. Click the *Email log* button to start drafting an email in Microsoft Outlook with the debugging log included.[1]

The last button on the Windows version is *Shortcuts*. It allows the creation of application shortcuts in the following locations: the desktop, the SendTo menu and the Start menu. DeLaTeXify cannot add a shortcut to the taskbar directly, but one can be added by dragging one from the other three locations. As mentioned above, dragging-and-dropping a LaTeX file

[1] This button was added to facilitate reporting errors and unsupported LaTeX commands to the DeLaTeXify developer.



(a) On Microsoft Windows 7.



(b) On Linux.

Figure 2: *DeLaTeXify GUI (Version 3.0 beta 3).*

on one of these shortcuts starts DeLaTeXify and fills the input field automatically. However, the shift key must be pressed when dropping a file on the taskbar; otherwise, Windows adds the file to the jump list of the taskbar shortcut rather than launch DeLaTeXify. Unfortunately, the jump list does not work with DeLaTeXify, as Windows sees it as a Python jump list rather than a DeLaTeXify jump list.

3.1.2 Debugging options

This section of the GUI provides various options that can help with debugging when creating or updating replacement rules:

1. **nodefault:** Deactivate the default rules that DeLaTeXify uses for LaTeX commands that have no explicit rules defined;[2]
2. **nolocal:** Omit local rules defined in a *local_delatexify.py* file (see Annex A);
3. **steps:** Print the converted text (in its current state) to the debugging log after applying each replacement rule;
4. **trace:** List to the debugging log the search patterns and replacement rules as they are run;
5. **times:** Save the compilation and run times for the search patterns and replacement rules to a Comma-Separated Values (CSV) file with the same root name as the output file; and
6. **re:** Use the *re* module even if the *regex* module is available.

These options are described in more detail in Section 4.3.

3.1.3 Debugging log

DeLaTeXify writes debugging and diagnostic information to this section:

1. The path to the local rules file (*local_delatexify.py*) if one is used (see Annex A);
2. The name of the LaTeX log file used to identify document classes and packages (or the word *missing* if no log was found);
3. The path to files inserted into the main LaTeX document using the **\input**, **\include** and **\bibliography** commands;
4. The list of LaTeX commands left at the end of the conversion, if any;
5. Converted text after the application of each individual replacement rule, if the steps option is specified;

[2] DeLaTeXify currently has four default rules. They remove **\begin**, **\end** and argument-less commands from the text, and replace one-argument commands with the content of their argument.

6. Search patterns and replacement rules as they are executed, if the trace option is specified;
7. Any syntax error encountered in regular expressions; and
8. Any Python error that occurs during program execution.

3.2 Command-Line Interface (CLI)

All of the DeLaTeXify functionality, aside from the Windows-specific buttons (*Check*, *Email log* and *Shortcuts*), can be accessed from the command line. On Windows, it can be run by typing `python delatexify.py` followed by the desired command-line arguments. On Linux, `delatexify.py` can be used as the command name and there is no need to type `python` before it. If no command-line argument is specified, DeLaTeXify starts in GUI mode. If the `-h` option is specified, a help message is printed. (See Figure 3.) The `--version` option prints out the version number of DeLaTeXify. All other arguments correspond to their GUI equivalent.

3.3 Usage notes and limitations

This section provides a few usage guidelines for DeLaTeXify and summarizes its main limitations.

3.3.1 LaTeX log file or `\usepackage` commands

LaTeX packages often depend on other packages and load them upon initialization. The same is also true of some document classes. LaTeX allows commands from classes and packages that are loaded indirectly by another one to be used even if they were not loaded explicitly using the **`\documentclass`** and **`\usepackage`** commands. If the LaTeX document has been compiled, DeLaTeXify uses the log file produced by LaTeX to identify those classes and packages. However, if the LaTeX log file is not available, DeLaTeXify only loads rules for the document class specified using **`\documentclass`** and for packages loaded using **`\usepackage`**.

For instance, the TikZ package loads the xcolor package, which provides colour-related commands such as `\definecolor`. This allows `\definecolor` to be used in documents that have **`\usepackage{tikz}`** even if **`\usepackage{xcolor}`** is not specified. If the LaTeX log file is available, DeLaTeXify realizes that xcolor is loaded when **`\usepackage{tikz}`** is used and applies the rule for `\definecolor` even if **`\usepackage{xcolor}`** is not specified in the document. However, if the LaTeX log file is not available, the DeLaTeXify rule for `\definecolor` is only applied if **`\usepackage{xcolor}`** appears explicitly in the document. In order for DeLaTeXify to work properly, it is thus important to process the LaTeX file before running DeLaTeXify or to include an explicit **`\usepackage`** command for all packages that provide commands used in the document.

```
usage: delatexify.py [-h] [--outfile OUTFILE] [--gui] [--version]
                     [--nodefault] [--nolocal] [--steps] [--trace]
                     [--times] [--re]
                     [INFILE.tex]

Convert LaTeX file to plain text in order to reduce the number of false
positives when checking grammar with Microsoft Word or other software.

positional arguments:
  INFILE.tex    input file

optional arguments:
  -h, --help    show this help message and exit
  --outfile OUTFILE, -o OUTFILE
                pattern for name of output file without extension, with
                %r standing for root name of input file (name without
                .tex extension); .txt extension added automatically;
                default: %r
  --gui         launch in GUI mode
  --version     print out version number and exit
  --nodefault   omit default rules, to help debug command-specific rules
  --nolocal     omit rules from local_delatexify.py
  --steps       print text to standard error after applying each rule, to
                help debug interactions between them
  --trace       list patterns and rules to standard error as they are
                run, to help identify the source of catastrophic
                backtracking
  --times       save compilation and run times of regular expressions to
                OUTFILE.csv
  --re          use standard re module even if third-party regex module
                is available
```

Figure 3: *CLI help message of DeLaTeXify (Version 3.0 beta 3).*

3.3.2 Default rule confusion

As mentioned in footnote 2 on page 7 and in Section 4.3.1, DeLaTeXify provides default rules to process environments, no-argument commands and one-argument commands for which no explicit rule is specified. This helps reduce the number of explicit rules required. However, the default rules can get confused when curly, round or square brackets follow a command (even if the opening bracket is preceded by spaces), as the bracketed content is mistakenly interpreted as an additional command argument. After **\begin**, it is removed with the rest of the **\begin** command. After one-argument commands, it prevents the application of the default rule. After no-argument commands, it leads to the erroneous application of the default rule for one-argument commands. These problems can be mitigated in a number of ways:

1. Creating an explicit rule for the affected command or environment;
2. Using an explicit space—either a tilde (`~`) or a backslash-space pair (`\ `)—between the last argument of the command and the following bracketed content; or
3. Wrapping the command in curly brackets.

Because of this issue and in order to reduce the amount of document customization required, DeLaTeXify provides explicit rules for a number of common one-argument commands such as **`\mbox`**, **`\emph`** and **`\textbf`**.

3.3.3 Stray parentheses with *re* module

When running the *re* module, certain combinations of LaTeX commands lead to stray parentheses being left in the document when there are more than two levels of curly braces involved. More specifically, this happens when using a **`\footnote`**, **`\footnotetext`** or **`\marginpar`** command in **`\newcommand`**, **`\renewcommand`** or **`\providecommand`**. This issue does not arise with the *regex* module. The problem with the *re* module can be mitigated by placing the command definition between **`\makeatletter`** and **`\makeatother`**, as DeLaTeXify discards all text between those two commands.

3.3.4 Percent signs in `\verb` commands

Percent signs (`%`) in **`\verb`** commands are not processed properly. They are interpreted as starting a comment. A workaround is to replace **`\verb`** by the **`\lstinline`** command of the listings package with the mathescape option set to true. With this option, LaTeX commands can be used between pairs of dollar signs in **`\lstinline`** and a command can be defined to represent the percent sign.

3.3.5 Escaped curly brackets

DeLaTeXify makes no provision to process escaped curly brackets properly. They are interpreted as regular curly brackets and processed as such.

4 Replacement rules

As mentioned in the introduction, the text from LaTeX documents is extracted through the application of text replacement rules. The set of rules currently implemented in DeLaTeXify covers LaTeX commands used over the years by the author and users of the software. Additional rules can be defined as required. These rules can be stored in three possible locations:

1. **Document:** Rules for LaTeX commands that are specific to a document should be stored in the document itself, as described in this section;
2. **local_delatexify.py:** Rules for LaTeX commands that a user defines locally but uses in multiple documents may optionally be stored in a *local_delatexify.py* file (see Annex A); and
3. **delatexify.py:** Rules for LaTeX commands that come from standard LaTeX packages should be sent to the DeLaTeXify developer for inclusion into the main tool.

With a few exceptions, rules defined in a LaTeX document are applied before those defined in *local_delatexify.py* and *delatexify.py* and those in *local_delatexify.py* are applied before those defined in *delatexify.py*. (More information about the order of rule application is provided in Annex B.) This allows users to override rules provided with DeLaTeXify with their own rules, if desired. It also lets them write rules in terms of those provided with DeLaTeXify, which can help avoid reinventing or duplicating complex rules and ensure consistency if standard DeLaTeXify rules are modified in future versions.[3]

The syntax of text replacement rules is described in Section 4.1, examples are provided in Section 4.2 and the debugging features of DeLaTeXify are presented in Section 4.3. The reader is assumed to have a basic knowledge of regular expressions.

4.1 Syntax

Replacement rules specified in LaTeX documents are written in comments so not to interfere with file compilation. Each rule is written as "`% Rule(pattern, replacement, flags)`", where `%` starts the comment (and must be the first character of the line), `pattern` is the search pattern, `replacement` is the replacement specification and `flags` is an optional set of flags that specify rule options. The `pattern`, `replacement` and `flags` arguments are described in Sections 4.1.1 to 4.1.3. Each rule can be specified over multiple lines if required; in that case, each continuation line must also start with `%`.

[3] An example of this is provided in Figure C.1 at the beginning of Annex C. It shows how the rule for the `\eqref` command is expressed in terms of the `\ref` command.

4.1.1 Search pattern

The search pattern is a regular expression. It uses the syntax defined in the documentation for the Python *re* module [5].[4] In addition, it also supports possessive quantifiers and atomic groups from the Python *regex* module [4], as they can reduce the risk of catastrophic backtracking.[5] When the *re* module is used, they are automatically replaced with greedy quantifiers and non-capturing groups, respectively.

DeLaTeXify defines a number of LaTeX extensions to the usual regular expression syntax. All of these extensions start with the percent character (`%`):

1. The `%c`, `%r` and `%s` strings in search patterns are replaced with regular expression patterns that match arguments of LaTeX commands:
   (a) `%c` matches a pair of curly brackets with arbitrary content in between;
   (b) `%r` is like `%c`, but for round brackets (parentheses); and
   (c) `%s` is like `%c`, but for square brackets;
2. The `%h`, `%n` and `%w` strings in search patterns are replaced by regular expression patterns that match optional white space:
   (a) `%h` matches an arbitrary amount of horizontal white space (space or tab), including none;
   (b) `%n` is similar to `%h`, but may also include at most one newline character; and
   (c) `%w` is similar to `%n`, but may include an arbitrary number of newline characters.

In the search pattern, arguments of LaTeX commands can be marked as optional by putting a question mark (`?`) after the corresponding `%c`, `%r` or `%s`.

The in-bracket content matched by `%c`, `%r` and `%s` is captured so that it can be referenced in replacement specifications in the same way as the text captured by pairs of parentheses. In other words, `%c`, `%r` and `%s` define capturing groups. These capturing groups are indexed from 1:

1. The first `%c`, `%r`, `%s` or pair of parentheses defines capture group number 1;
2. The second `%c`, `%r`, `%s` or pair of parentheses defines capture group number 2;
3. ...

The text matched by `%h`, `%n` and `%w` is not captured.

[4] Regular expression syntax varies slightly from programming language to programming language.
[5] Catastrophic backtracking happens when a regular expression does not match the text, but is written in such a way that it takes an extremely long time for the search engine to determine that.

4.1.2 Replacement specification

For every part of the text that matches the search pattern, DeLaTeXify replaces the matched text with the one specified by the replacement string.[6] In the replacement string:

1. `\1` refers to the first capture group of the search pattern (`%c`, `%r`, `%s` or pair of parentheses);
2. `\2` refers to the second capture group;
3. ...

Also, `\n` refers to a newline character.

4.1.3 Flags

The optional `flags` argument can be used to set the following two options:

1. **ITERATIVE:** The rule is applied iteratively until it no longer matches; and
2. **DEFAULT:** The rule is deactivated if DeLaTeXify is run with the nodefault option specified.

The two options can be specified concurrently by setting `flags` to `ITERATIVE|DEFAULT`, where `|` is the bitwise OR operator.

4.2 Examples

The examples provided in this section are extracted from *delatexify.py*, where they are not prepended by the LaTeX comment character (`%`). However, the `%` is included here to show what the rules would look like if they were defined in a document.

4.2.1 Example 1: \label

```
% Rule(r'\\label%c', '')
```

In this rule, the search pattern is `r'\\label%c'` and the replacement specification is `''`. It replaces all `\label` commands with an empty string—thus removing them from the text entirely. The search pattern is prepended by `r` to indicate that it is a raw string; this prevents Python from interpreting the backslashes as special characters. This is not needed for the replacement specification, because it does not contain any backslash character. The backslash is also a special character in regular expressions. In order to represent a single backslash in LaTeX, a double backslash must therefore be written in the regular expression. That is why the search pattern starts with a double backslash.

[6] For rules defined in LaTeX documents, replacement specifications must be strings. For rules defined in a local repository (*local_delatexify.py*), they can also be functions or classes. (See Annex A.)

4.2.2 Example 2: \ref

```
% Rule(r'\\ref%c', 'X')
```

In this rule, the search pattern is `r'\\ref%c'` and the replacement specification is `'X'`. Since the replacement specification does not refer to any capture group, the replacement text is constant. Each occurrence of the `'\ref'` command in the text is simply replaced by the letter X, to represent the fact that its value is unknown. Since the actual reference number has no impact on the grammar, there would be no benefit in determining it and showing it in the text.

4.2.3 Example 3: sectioning commands

```
% Rule(r"""\\(?:part|chapter|section|subsection|subsubsection
%                    |paragraph|subparagraph)\*?%s?%c""",
%       r'\n\1\n\n\2\n')
```

In this rule, the search pattern is written as a multi-line string using triple quotes. Since search patterns in DeLaTeXify are interpreted using the verbose option [5], the newline character after the first line of the search pattern and the spaces at the beginning of its second line are ignored. The vertical bar token `|` signifies alternation; it means that the search pattern matches any of the sectioning commands, such as **`\part`**, **`\chapter`** and **`\section`**. The alternation is encapsulated in a non-capturing group, denoted by `(?: )`, to indicate that the characters outside of the group are not part of the alternation. The alternation group is followed by `\*?` to indicate that the sectioning commands may be followed by a star. This way, the search pattern matches both the regular and starred versions of the sectioning commands. The optional star is followed by `%s?` and `%c`, which respectively match the optional and mandatory arguments of the sectioning commands.

The replacement specification refers to both arguments. It replaces the sectioning command by its two arguments separated by an empty line, the pair of which is preceded and followed by newline characters. As long as the sectioning command is on a line of its own, this makes each of the two title versions bracketed by empty lines—thus ensuring that Word sees each of them as a single paragraph when checking grammar.

4.2.4 Example 4: footnotes

```
% Rule(r'(?s)\\footnote(?:text)?%s?%c(.*?)\n%h\n',
%       r'\3 (\2)\n\n', ITERATIVE)
```

This iterative rule matches the **`\footnote`** and **`\footnotetext`** commands. The `(?s)` at the beginning of the search pattern indicates that the period token `.` matches all characters including newlines.[7] The `(?:text)?` part of the search pattern indicates that `text` is optional, which is what allows the search pattern to match both **`\footnote`** and **`\footnotetext`**. Then, `%s?` and `%c` respectively match the optional and mandatory arguments of the commands. Finally, `(.*?)\n%h\n` matches the rest of the paragraph after the end of the footnote

[7] Without `(?s)`, the period token `.` would match all characters except newlines.

up to the next empty line and assigns it to the third capturing group, denoted by the pair of parentheses.

The replacement specification puts the footnote text in parentheses at the end of the paragraph. It also drops the optional argument and replaces the empty line matched by the search pattern. When a paragraph has more than one footnote, applying the rule once moves the first footnote to the end of the paragraph, but leaves the other ones intact. The `ITERATIVE` option added to the rule makes DeLaTeXify apply it until it no longer matches. This way, all footnotes are individually moved to the end of their corresponding paragraph in order of appearance.

4.2.5 Example 5: \subcaption

```
% Rule(r'\\subcaption%s?%c', r'\\caption[\1]{\2}')
```

In this rule, the **\subcaption** command from the subcaption package is replaced by the core **\caption** command. DeLaTeXify then applies the rule for the `caption` command. This works because rules for core LaTeX commands are applied after rules for packages (see Annex B). The main advantage of this approach is that it ensures formatting consistency between the two commands (**\caption** and **\subcaption**). If the rule for **\caption** is ever modified, the one for **\subcaption** will be benefit from the modification automatically. Another option would be to modify the rule for **\caption** so it applies to both commands. However, **\subcaption** is provided by a package, and it is cleaner to load the applicable rule only when needed in case another package provided a syntactically incompatible version of the **\subcaption** command.

4.3 Debugging options

As mentioned in Section 3.1.2, DeLaTeXify provides a number of options to help debug rules when they do not produce the anticipated result. They are described in more detail in this section. Annex C provides additional debugging information for advanced users.

4.3.1 nodefault option

DeLaTeXify has four default rules that are applied to commands left after applying the command-specific rules:

1. Remove argument-less commands;
2. Remove **\begin** commands;
3. Remove **\end** commands; and
4. Replace one-argument commands, except **\begin** and **\end**, with the content of their argument.

Sometimes, default rules can mask errors in other rules. The nodefault option deactivates default rules in order to allow the user to see more clearly how well the command-specific rules work.

4.3.2 nolocal option

DeLaTeXify allows the user to create a repository of local rules in a *local_delatexify.py* file (see Annex A). The nolocal option deactivates these rules, to help diagnose if a problem is caused by local rules.

4.3.3 steps option

Sometimes, problems with the text conversion arise because of interactions between multiple rules rather than any single one. In those cases, it is useful to see how each rule changes the text in order to analyze those interactions. When the steps option is specified, DeLaTeXify outputs the text in its current state to the debugging log after every change along with the rule that produced the change—thus allowing the user to step through the text conversion. In order to simplify the analysis of the output produced by this option, it is better to use it with short text excerpts rather than long documents.

An example of the output produced by this option is shown in Figure 4. The original LaTeX document is listed at the top of the output, in lines 1 to 4 in this case. The following lines describe each rule that matched the document with the incremental output. For instance, lines 5 to 12 correspond to the first rule:

- **Line 5:** separator between the original document and the first matching rule;
- **Line 6:** file, line and function where the rule is defined, followed by the rule definition;
- **Line 7:** number of times that the rule matched the document;
- **Line 8:** separator between the description of the rule and the output resulting from its application; and
- **Lines 9 to 12:** state of text after the rule is applied.

In this case, four rules matched the document:

- The first rule (line 6) removed the **`\documentclass`** command;
- The second and third rules (lines 14 and 21) were default rules that removed all remaining **`\begin`** and **`\end`** commands (there was only one of each); and
- The fourth rule (line 27) removed the remaining empty line at the beginning of the document.

```
\documentclass{article}
\begin{document}
Hello world!
\end{document}
================================================================================
delatexify.py, line 2203, rules_core: Rule(r'\\documentclass%s?%c', '')
Matches: 1
--------------------------------------------------------------------------------

\begin{document}
Hello world!
\end{document}
================================================================================
delatexify.py, line 2317, rules_cleanup: Rule(r'\\begin%c(?:%c|%r|%s)*+%n', '', DEFAULT)
Matches: 1
--------------------------------------------------------------------------------

Hello world!
\end{document}
================================================================================
delatexify.py, line 2318, rules_cleanup: Rule(r'\\end%c%n', '', DEFAULT)
Matches: 1
--------------------------------------------------------------------------------

Hello world!
================================================================================
delatexify.py, line 2330, rules_cleanup: Rule(r'\A\n++', '')
Matches: 1
--------------------------------------------------------------------------------
Hello world!
```

Figure 4: *Sample output from the steps option.*

4.3.4 trace option

When the trace option is selected, DeLaTeXify prints each search pattern and replacement rule to the debugging log before running it. This allows the identification of the culprit when a search pattern or replacement rule faces catastrophic backtracking. Lines of the trace log are similar to lines 6, 14, 21 and 27 of Figure 4. (Search patterns appear as `Pattern(pattern)`, where `pattern` is a regular expression that uses the syntax described in Section 4.1.1.) However, a trace line is produced each time that a search pattern or replacement rule is applied, even if it does not match. For the example shown in Figure 4, the trace contains 1252 lines.

4.3.5 times option

When the times option is selected, DeLaTeXify saves the compilation and run times for the search patterns and replacement rules to a CSV file. This allows the user to see which rules are being applied and to assess their run time efficiency. The CSV file contains the following columns:

1. **File:** name of file where rule is defined (delatexify.py, local_delatexify.py or the LaTeX document);
2. **Line:** number of line where rule is defined;
3. **Scope:** name of function or class where rule is defined (empty for rules defined in LaTeX document);
4. **Compilation Time:** compilation time in seconds;
5. **Run Time:** total run time in seconds, summed over all runs;
6. **Run Count:** number of times that search pattern or replacement rule was run;
7. **Matches:** number of times that search pattern or replacement rule matched the text, summed over all runs; and
8. **Object:** definition of search pattern or replacement rule.

On Windows, the compilation and run times correspond to clock time; on Linux, they correspond to Central Processing Unit (CPU) time.

4.3.6 re option

By default, DeLaTeXify uses the *regex* module whenever available because it leads to a faster execution. The re option forces it to use the *re* module, for testing purposes.

5 Conclusion

DeLaTeXify converts LaTeX documents to plain text, so that their grammar can be verified using tools such as Microsoft Word. It is based on text replacement rules defined using regular expressions. Rules are provided for many LaTeX commands already, but additional rules can be defined by users on a per-document basis for commands that are defined locally or are not yet supported by DeLaTeXify. Such rules can also be used to override those provided with DeLaTeXify in order to customize the output. While not mandatory, users can create local rule repositories for custom commands used in multiple documents. If users create rules for additional commands from the standard LaTeX classes or packages or the standard BibTeX styles, they should forward them to the DeLaTeXify developer for inclusion into the base product, so that others can benefit from them. DeLaTeXify is expected to support an increasing number of LaTeX commands and packages as its user base develops.

References

[1] TeX4ht (online), TeX Users Group, http://tug.org/tex4ht/ (Access Date: January 2020).

[2] Pandoc: a universal document converter (online), John MacFarlane, https://pandoc.org/ (Access Date: January 2020).

[3] Regular-Expressions.info (online), Jan Goyvaerts, https://www.regular-expressions.info/ (Access Date: January 2020).

[4] regex: alternative regular expression module, to replace re (online), Matthew Barnett, https://pypi.org/project/regex/ (Access Date: January 2020).

[5] re – Regular expression operations (online), Python Software Foundation, https://docs.python.org/3/library/re.html (Access Date: January 2020).

[6] tkinter – Python interface to Tcl/Tk (online), Python Software Foundation, https://docs.python.org/3/library/tkinter.html (Access Date: January 2020).

Annex A Local rules

It is generally sufficient to keep custom replacement rules in the LaTeX document itself. However, if a user would like to use the same custom replacement rules for multiple documents, they can be placed in a local rules file named *local_delatexify.py* rather than in the documents. This file should be placed in the same directory as *delatexify.py*. Figure A.1 provides a fictitious example of such a file.

```
1  import delatexify
2  from delatexify import Rule, RuleList
3
4  def rules_class_drdc_report():
5      return Rule(r'\\projectnumber%1', '')
6
7  def rules_package_foobar():
8      return RuleList([
9          Rule(r'\\foo%c%c', r'\1'),
10         Rule(r'\\bar%c%c', r'\2')
11     ])
12
13 def rules_style_drdc_custom():
14     return delatexify.rules_style_drdc()
```

Figure A.1: *Fictitious local_delatexify.py file.*

The first line of *local_delatexify.py* imports *delatexify.py*, and the second line provides shortcuts for the `Rule` and `RuleList` classes. These shortcuts allow rules to be written exactly the same in the two files and to simplify the cutting-and-pasting of rules between them. As discussed in Section 4, the `Rule` class is used to define conversion rules.[8] The `RuleList` class is used to group these rules into lists. `RuleList` objects have the same interface as Python lists and as `Rule` objects. They are defined as `RuleList(rules, flag)`, where `rules` is a list of rules and where `flag` is an optional `ITERATIVE` flag as described in Section 4.1.3.[9] When the `ITERATIVE` flag is specified, the list of rules is repeated until none of them match.

The rest of *local_delatexify.py* is composed of functions that return either individual rules or lists of rules. The names of these functions all start with `rules_`. There are three main categories of such functions:

1. **`rules_class_X`:** Return rules for documents of class X;
2. **`rules_package_X`:** Return rules for commands of package X; and

[8] In *delatexify.py* and *local_delatexify.py*, rule specifications are not preceded by the LaTeX comment character (%).
[9] The `DEFAULT` flag does not apply to rule lists.

3. **`rules_style_X:`** Return rules for BibTeX style X.[10]

When class, package or style names contain hyphens or periods, they must be replaced with underscores in the function names. For instance, in Figure A.1, `rules_class_drdc_report` returns rules for the drdc-report document class, `rules_package_foobar` returns rules for the fictitious foobar package and `rules_style_drdc_custom` returns rules for a custom version of the Defence Research and Development Canada (DRDC) BibTeX style called drdc-custom.

In addition to these three categories of `rules_` function, *local_delatexify.py* can also contain the following functions:

1. **`rules_setup:`** Return rules that are executed before those defined in the LaTeX document (Step 1 in Annex B);
2. **`rules_core:`** Return rules for core LaTeX commands (Step 2e in Annex B);
3. **`rules_cleanup_braces:`** Return default rules for LaTeX commands for which no explicit rules were found and to remove braces that are not part of commands (Step 3 in Annex B); and
4. **`rules_cleanup:`** Return rules to process remaining LaTeX environments, symbols, punctuation marks and spaces (Step 4 in Annex B).

Rules in *delatexify.py* are defined using the same function names as in *local_delatexify.py*. If a `rules_` function is present in both files, the rules returned by the function in *delatexify.py* are applied immediately after those returned by the one in *local_delatexify.py*.

The functions shown in Figure A.1 illustrate three different ways in which a local rule file can customize the rule set:

1. **Replace an existing rule:** The rule returned by the `rules_class_drdc_report` function in *local_delatexify.py* is executed before those returned by the function of the same name in *delatexify.py*, which means that the standard rule for `\projectnumber` is effectively overshadowed by the local one;
2. **Define rules for additional commands:** The `rules_package_foobar` defines rules for two commands of the fictitious foobar package; and
3. **Indicate that a rule list defined for one class, package or style should be used for another:** By returning the rule list defined in *delatexify.py* for the standard DRDC BibTeX style, `rules_style_drdc_custom` specifies that these rules should also be used for the locally defined drdc-custom style.

[10] Some BibTeX styles insert LaTeX commands into the bibliography.

For rules defined in LaTeX documents, the replacement text is always specified using a string. There is however more flexibility for rules defined in *local_delatexify.py*. Indeed, both in *delatexify.py* and *local_delatexify.py*, the replacement can be specified using any of the three following means:

1. **String:** As discussed in Section 4.1.2;
2. **Function:** See documentation of re.sub function [5]; or
3. **Class of function object:** Similar to using a function, but the replacement text can vary based on the matching order.[11]

[11] Such a function object is used for instance by the rules that replace `%c`, `%r` and `%s` with search patterns for command arguments. (See Annex C.)

Annex B Order of rule application

DeLaTeXify applies rules in the following order:

1. Setup rules:
   (a) Insert external files imported using **\include**, **\input** and **\bibliography** commands;
   (b) Remove comments;
   (c) Remove lines between **\makeatletter** and **\makeatother**; and
   (d) Convert symbols, punctuation marks, tabs and accented letters;
2. Main rules:
   (a) Rules defined in the LaTeX document;
   (b) Rules specific to the document class;
   (c) Rules specific to each package, in the order in which the packages are loaded into the LaTeX document;
   (d) Rules specific to the bibliographic style; and
   (e) Rules for core LaTeX commands (aside from those processed by the setup and cleanup rules);
3. Brace-cleanup rules:
   (a) Replace one-argument commands with the content of their argument; and
   (b) Remove curly brackets that are not part of a command; and
4. Final cleanup rules:
   (a) Remove remaining **\begin**, **\end** and argument-less commands;
   (b) Rules for explicit spaces, such as `~`; and
   (c) Wrap lines and remove all superfluous spaces left after the text conversion.

At each step, rules defined in *local_delatexify.py*—if present—are executed before those provided by *delatexify.py*. (See Annex A for more information.)

When using the *regex* module, DeLaTeXify only needs to apply each rule once. It can do so because, by using recursive patterns [4], it can identify command arguments irrespective of the number of balanced bracket levels in each of them.[12] However, the *re* module does not support recursive patterns. Correspondingly, the search patterns for command arguments when using that module are limited in the number of balanced bracket levels that are allowed

[12] For each argument, unbalanced brackets of the type surrounding the argument are forbidden. For instance, a curly-bracket argument can contain unbalanced parentheses and square brackets, but not unbalanced curly brackets.

in arguments: they only allow one level in addition to the one surrounding the argument.[13] If more than one level of brackets is present in an argument, the pattern does not match and the rule is not applied. For this reason, when multiple levels of LaTeX commands are nested, DeLaTeXify must process the commands from the inside out and needs to apply the rules multiple times in order to process all of them. It does so by looping over the rules multiple times until they no longer match. More precisely, it loops over the rules in the following manner:

- Step 1 (setup rules) is executed once;
- Step 2 (main rules) is executed repeatedly until none of its rules matches the text;
- If any of the brace cleanup rules in step 3 matches the text, the algorithm goes back to step 2; and
- Step 4 (final cleanup rules) is executed once.

[13] For each argument, the number of bracket levels is limited only for the type of brackets surrounding the argument. For instance, a curly-bracket argument can contain an unlimited number of levels of parentheses or square brackets, but only one level of curly brackets when using the *re* module. With the *regex* module, the number of bracket levels allowed is practically unlimited.

Annex C Syntax errors

When creating new rules, syntax errors may at times be encountered in search patterns and replacement specifications. When this happens, DeLaTeXify writes an error message to the debugging log detailing the problem. To illustrate this mechanism, let us consider a short (and incomplete) LaTeX file called *example.tex* listed in Figure C.1. The first line is a copy of the rule provided in DeLaTeXify for the **\eqref** command of the amsmath package. It does the same thing as the core **\ref** command, but in parentheses in order to mimic the actual behaviour of **\eqref**. When this rule is applied to the second line, it yields **(X)**.

```
1 % Rule(r'\\eqref%c', r'(\\ref{\1})')
2 \eqref{eq}
```

Figure C.1: *Baseline file to illustrate syntax errors.*

If the replacement rule referenced the non-existing capturing group 2 in the replacement specification, the regular expression engine would raise an invalid capturing group error. DeLaTeXify would report the location of the rule that generated the error and the error message from the regular expression engine, as shown in the first line of Figure C.2.[14] The second line of the report contains the faulty rule definition.

```
1 regex error (example.tex, line 1, ): invalid group reference
2 Rule(r'\\eqref%c', r'(\\ref{\2})')
```

Figure C.2: *Syntax error if an invalid capturing group is referenced.*

If instead of an invalid reference in the replacement specification, the search pattern had a stray closing parenthesis between **eq** and **ref**, the regular expression engine would raise an unbalanced parenthesis error and DeLaTeXify would produce the error report in Figure C.3. The first line is similar to the first error situation, but this time the engine indicates the position in the regular expression where the error was detected. In this case, the unbalanced parenthesis is at character 18 (indexed from zero). The second line of the error message contains the regular expression up to the error point. The third line highlights the position of the error by a vertical line preceded by hyphens. The part of the regular expression that comes after the error point is written in lines 4 to 24. (The comments and indentation in the regular expression are only used to help users understand the search pattern. They are ignored by the regular expression engine, because DeLaTeXify uses the verbose option.)

The search pattern displayed by DeLaTeXify in Figure C.3 is a lot more complex than the one shown in Figures C.1 and C.2. The reason is that Figures C.1 and C.2 show the search

[14] If the rule were defined in *delatexify.py* or *local_delatexify.py*, the name of the class or function where the definition is located would be listed after the line number.

```
regex error (example.tex, line 1, ): unbalanced parenthesis at position 18 (line 1, column 19)
(?<!(?<!\\)\\)\\eq
------------------|
                   )ref
                                                # CURLY-BRACKET ARGUMENT
        (?:                                     # Non-capt group for quantifiers
            [\ \t]*\n?[\ \t]*                   # Drop white space (one \n max)
            {                                   # Opening bracket
                (?=                             # Lookahead to ensure atomicity
                    (?P<c1>                     # Start capturing group
                        (?:                     # Non-capt. group for alternative:
                            [^{}]+              # Non-brackets
                            |                   # Or
                            (?:                 # Balanced brackets
                                {               # Opening bracket
                                    [^{}]*      # No bracket
                                }               # Closing bracket
                            )                   #
                        )*                      # Capture as much as possible
                    )                           # Stop capturing group
                )                               #
                (?P=c1)                         # Consume text matched by lookahead
            }                                   # Closing bracket
        )                                       # End non-capt group
```

Figure C.3: *Syntax error if unbalanced parentheses are present.*

pattern as defined in DeLaTeXify or by the user, whereas Figure C.3 shows it as seen by the regular expression engine. As mentioned in Section 4.1.1, DeLaTeXify replaces the `%c` appearing in the search pattern of the **`\eqref`** rule by a regular expression that matches a LaTeX argument in curly brackets—as show in lines 5 to 24 of Figure C.3. The version shown here is for the *re* module; the one for the *regex* module is slightly different. DeLaTeXify also prepends the search pattern with `(?<!(?<!\\)\\)`, to ensure that it matches **`\eqref`** and **`\\eqref`**, but not **`\\eqref`**.[15] While not applicable here, when a search pattern ends with an argument-less LaTeX command, DeLaTeXify appends a regular expression that swallows non-newline white space after the command and ensures that the pattern does not match the beginning of longer commands—for instance, to prevent a pattern for **`\abc`** from matching **`\abcd`**.

[15] In LaTeX, **`\\eqref`** stands for a newline character followed by **`eqref`** rather than the **`\eqref`** command. While **`eqref`** is not likely to appear on its own in typical LaTeX documents, this could be an issue with commands that have more common names such **`\and`**.

Abbreviations, acronyms and initialisms

CLI Command-Line Interface

CPU Central Processing Unit

CSV Comma-Separated Values

DRDC Defence Research and Development Canada

GUI Graphical User Interface

This page intentionally left blank.

DOCUMENT CONTROL DATA

*Security markings for the title, abstract and keywords must be entered when the document is sensitive.

1. ORIGINATOR (The name and address of the organization preparing the document. A DRDC Centre sponsoring a contractor's report, or a tasking agency, is entered in Section 8.)

DRDC – Centre for Operational Research and Analysis
NDHQ Carling, 60 Moodie Drive, Building 7S.2, Ottawa ON K1A 0K2, Canada

2a. SECURITY MARKING (Overall security marking of the document, including supplemental markings if applicable.)

CAN UNCLASSIFIED

2b. CONTROLLED GOODS

NON-CONTROLLED GOODS
DMC A

3. TITLE (The document title and subtitle as indicated on the title page.)

DeLaTeXify: Grammar-Checking Enabler for LaTeX: User Manual

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used. Use semi-colon as delimiter.)

Guillouzic, S.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total pages, including Annexes, excluding DCD, covering and verso pages.)	6b. NO. OF REFS (Total cited in document.)
October 2021	35	6

7. DOCUMENT CATEGORY (e.g., Scientific Report, Contract Report, Scientific Letter)

Reference Document

8. SPONSORING CENTRE (The name and address of the department project or laboratory sponsoring the research and development.)

DRDC – Centre for Operational Research and Analysis
NDHQ Carling, 60 Moodie Drive, Building 7S.2, Ottawa ON K1A 0K2, Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable contract number under which the document was written.)
99 – CG&S – Other	

10a. DRDC PUBLICATION NUMBER	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned to this document either by the originator or by the sponsor.)
DRDC-RDDC-2021-D076	

11a. FUTURE DISTRIBUTION WITHIN CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

Public release

11b. FUTURE DISTRIBUTION OUTSIDE CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

Public release

12. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Use semi-colon as a delimiter.)

LaTeX; grammar; user manual; regular expression; Python

13. ABSTRACT/RÉSUMÉ (When available in the document, the French version of the abstract must be included here.)

DeLaTeXify is a Python-based software that converts LaTeX documents to plain text, so that their grammar can be verified using tools such as Microsoft Word. It is based on text replacement rules defined using regular expressions. Users can define additional rules to process commands that are not yet supported by DeLaTeXify or are defined locally. They can also override the rules provided with DeLaTeXify if desired. The user manual provides installation and usage instructions, as well as information about how to define and debug new text replacement rules.

DeLaTeXify est un logiciel programmé en langage Python qui convertit les documents LaTeX en texte brut, de façon à ce que l'on puisse vérifier la grammaire avec des outils comme Microsoft Word. Il est fondé sur des règles de remplacement de texte définies à l'aide d'expressions régulières. Les utilisateurs peuvent définir des règles additionnelles pour traiter les commandes qui ne sont pas encore prises en charge par DeLaTeXify ou qui sont définies localement. Ils peuvent aussi remplacer les règles fournies avec DeLaTeXify au besoin. Le manuel de l'utilisateur contient les instructions d'installation et d'utilisation, ainsi que de l'information sur la façon de définir et de déboguer de nouvelles règles de remplacement de texte.