National Defence | Défense nationale

DEFENCE RESEARCH AND DEVELOPMENT CANADA (DRDC)
RECHERCHE ET DEVELOPPEMENT POUR LA DÉFENSE CANADA (RDDC)

ERRERS 3.2 (Enhanced Review of Reports via Extraction using Rule-based Substitutions)

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

Updates

February 2024

Revisions were made to the installation and usage sections to account for the following changes made since version 3.1.1:

1. On Microsoft Windows, validation by user of language variants detected by Microsoft Word when clicking *Check* button;
2. On Microsoft Windows, replacement of "Send To" shortcut by entry in "Open With" menu; and
3. Addition of *Delete* button to shortcut-creation window.

The installation section now describes the removal procedure. A few typographical errors were also fixed, and a sentence was added to describe the use of the tab key and space bar in the graphical user interface (GUI).

November 2023

Revisions were made to the introduction and installation sections to account for the fact that ERRERS has now been open sourced and can be installed from the Python Packaging Index without downloading the package manually (see Section 2.2.1). Screenshots were also updated to reflect the improved display of fonts in ERRERS 3.1.1 on Microsoft Windows.

September 2023

Revisions were made throughout the document, including the title, to reflect changes in the tool since DeLaTeXify 3.0 beta 3, in particular:

1. New name and icon;
2. Support for Apple macOS;
3. Discontinuation of support for Python 2.7 to 3.5;
4. Reorganization as a platform-independent Python package, which led to changes in the installation procedure;
5. Creation of application shortcuts on Apple macOS and Linux in addition to Microsoft Windows;
6. New GUI layout, with separate windows for help and options;
7. Buttons in GUI to copy extracted text and extraction log to clipboard;
8. Keyboard shortcuts for buttons, checkboxes, and input fields;
9. Identification of document class and packages from LaTeX log if available;
10. Automatic generation of rules for LaTeX commands, environments, and counters defined in document;

11. Addition of placeholder `%C` (for regular expressions) that can match non-bracketed LaTeX commands and single characters in addition to arbitrary content in curly brackets;
12. References to capture groups in replacement specifications must be done by name rather than index (names are generated automatically by default but can be specified manually);
13. Extraction is now split into seven phases: location, insertion, removal, setup, main, cleanup_braces, and cleanup;
14. Improved detection of catastrophic backtracking; and
15. More detailed logging.

The change log distributed with ERRERS provides an exhaustive list of the changes.

Abstract

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

Résumé

ERRERS est un logiciel programmé en langage Python qui extrait le texte de documents LaTeX de façon à ce que l'on puisse vérifier la grammaire et l'orthographe avec des outils comme Microsoft Word. Il est fondé sur des règles de substitution de texte définies à l'aide d'expressions régulières. Des règles sont fournies pour de nombreuses commandes LaTeX, et des règles supplémentaires sont créées automatiquement pour les commandes définies dans des documents. Les utilisateurs peuvent définir des règles additionnelles pour traiter les commandes qui ne sont pas encore prises en charge par ERRERS ou qui sont définies localement. Ils peuvent aussi remplacer les règles fournies avec ERRERS 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 substitution de texte.

Table of contents

Updates i
Abstract iii
Résumé iii
Table of contents iv
List of figures vii
Acknowledgements viii
1 Introduction 1
2 Installation 3
2.1 Software requirements 3
2.2 Installation procedure 3
2.2.1 Option 1 3
2.2.2 Option 2 4
2.3 Removal procedure 4
3 Usage 5
3.1 Graphical user interface (GUI) 5
3.1.1 Controls 5
3.1.2 Runtime options 7
3.1.2.1 Logging 7
3.1.2.2 Substitution rules 7
3.1.2.3 Regular expression module 7
3.1.3 Extraction log 8
3.2 Command-line interface (CLI) 8
3.3 Usage notes and limitations 8
3.3.1 LaTeX log file or **`\usepackage`** commands 10
3.3.2 Default rule confusion 10
3.3.3 Differences between *regex* and *re* modules 10

4 Substitution rules . . . 12
4.1 Syntax . . . 12
4.1.1 Search pattern . . . 13
4.1.2 Replacement specification . . . 14
4.1.3 Iterative . . . 14
4.1.4 Phase . . . 14
4.2 Examples . . . 14
4.2.1 Example 1: \label . . . 14
4.2.2 Example 2: \ref . . . 15
4.2.3 Example 3: sectioning commands . . . 15
4.2.4 Example 4: footnotes . . . 15
4.2.5 Example 5: \subcaption . . . 16
4.3 Runtime options . . . 16
4.3.1 No auto . . . 16
4.3.2 No default . . . 16
4.3.3 No local . . . 17
4.3.4 Patterns . . . 17
4.3.5 *re* module . . . 17
4.3.6 Steps . . . 17
4.3.7 Timeout . . . 19
4.3.8 Times . . . 19
4.3.9 Trace . . . 19
4.3.10 Verbose . . . 20
5 Conclusion . . . 21
References . . . 22
Annex A Local rules . . . 23
Annex B Order of rule application . . . 26

Annex C Syntax errors 27

Abbreviations, acronyms, and initialisms 30

List of figures

Figure 1: Example of text extraction with ERRERS 1
Figure 2: ERRERS GUI (Version 3.2) on Windows 10 6
Figure 3: Command-line arguments and options of ERRERS (Version 3.2) 9
Figure 4: Sample output from the *steps* option 18
Figure A.1: Fictitious `local.py` file 23
Figure C.1: Baseline file to illustrate syntax errors 27
Figure C.2: Syntax error if an invalid capturing group is referenced 27
Figure C.3: Syntax error if unbalanced parenthesis is present 28

Acknowledgements

The author thanks the following people for their contribution to the development of ERRERS:

- Patrick Dooley, Pierre-Luc Drouin, Fred Ma, Matthew MacLeod, Paul Melchin, and Stephen Okazawa for their help in brainstorming and choosing a name for the tool;
- Janice Lang for suggesting the original idea for the icon and Adison Rossiter for designing it; and
- Pierre-Luc Drouin, Joshua Goldman, Fred Ma, and Paul Melchin for their help with beta testing.

1 Introduction

While many text editors used with LaTeX can perform spellchecking, support for grammar checking is limited. ERRERS aims to fill that gap.[1] It extracts the text from LaTeX files and formats it 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], ERRERS 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 reference numbers are irrelevant for checking grammar.

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

```
\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 has only 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 has only one sentence.
```

(b) *Extracted text.*

Figure 1: *Example of text extraction with ERRERS.*

The extraction is performed by applying a set of substitution 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 ERRERS,

[1] The ERRERS acronym stands for Enhanced Review of Reports via Extraction using Rule-based Substitutions.

such substitution rules have been defined for the more common LaTeX commands, and rules are created automatically for commands defined in documents using the **`\newcommand`**, **`\newenvironment`**, and **`\def`** families of commands. More rules can be added as required to support additional commands.

The author of this Reference Document started developing ERRERS in the early-to-mid 2000s. Version 1 was simply a set of search-and-replace rules in Microsoft Visual Basic for Applications (VBA) [4]. Work on version 2 started in 2006. Implemented in Vimscript [5], it involved the replacement of search patterns with regular expressions, which allowed the use of more complex substitution rules. For version 3, ERRERS was ported to Python and many new features were added, such as a graphical user interface (GUI), better debugging support for substitution rules, and the selection of rules based on the document class and packages used. While versions 1 and 2 were used only by the author, version 3.0 was shared with Defence Research and Development Canada (DRDC) colleagues, and version 3.1 was released under the MIT license.[2]

Sections 2 and 3 of this manual respectively indicate how to install and run ERRERS. Section 3 is the only section required reading to start using the tool if it is already installed. Section 4 then describes how to customize ERRERS 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] Version 2 and version 3.0 were internally known as DeLaTeXify. The name ERRERS was chosen for version 3.1.

2 Installation

This section describes how to install ERRERS. Section 2.1 summarizes software requirements, while Sections 2.2 and 2.3 respectively go over the installation and removal procedures.

2.1 Software requirements

ERRERS 3.2 runs on Python 3.6 or more recent [6]. It was tested on Microsoft Windows 10 and 11, on Apple macOS 14, and on Debian GNU/Linux 12.[3] It processes regular expressions using the third-party *regex* module [8] if available, but defaults back to the standard *re* module [9] if not. While the *regex* module allows ERRERS to run faster and more robustly, the *re* module is present in all Python distributions and thus ensures portability. For the graphical user interface, ERRERS uses Tkinter [10], which is also a standard component of all Python distributions. The Python distribution provided by Apple on macOS includes an older version of the Tkinter GUI library with which ERRERS is not compatible, but it can still be used in command-line mode. To use ERRERS in GUI mode on that platform, an alternate version of Python must be installed, such as those available from the Python website [6]. On Windows, ERRERS also uses the *pywin32* package [11] to create some shortcuts and to launch the document review in Microsoft Word.

2.2 Installation procedure

ERRERS is distributed as a Python package, which requires a local Python installation. It is packaged as a platform-independent Python wheel, which is a standard format for distributing Python packages [12]. There are two main options for installing it, detailed in the following subsections.[4]

2.2.1 Option 1

The simplest way to install ERRERS on computers that have unrestricted access to the Internet is as follows:

1. Open a command prompt or shell from which Python can be run. For instance, with the Anaconda Python distribution on Windows, you would choose Programs ⇒ Anaconda3 ⇒ Anaconda Prompt in the Windows Start menu.

2. Type `python3 -m pip install errers` and press enter. On some systems, you may need to use `py` or `python` rather than `python3`. You may also need to use `pipx` rather than `pip`.

3. Type `errers --shortcuts` at the command prompt and press enter to start the shortcut-creation window.

4. In the shortcut-creation window, choose locations where you would like application shortcuts to be created and click “Create.” Available locations vary from one operating system to another:
   - **On Windows:** Desktop, “Open With” menu, and Start menu;
   - **On macOS:** User Applications folder, Launchpad, and “Open with” menu; and

[3] GNU stands for GNU’s Not Unix [7].

[4] Your local network administrator may offer other methods of installing ERRERS. They are not covered here, as they are outside the scope of this manual.

- **On Linux:** Desktop (or Home directory), Application menu, and "Open With" menu.

5. Copy shortcuts to other locations if desired. For instance, on Windows, the shortcut from the Desktop or Start Menu could be pinned to the taskbar; on macOS, the Application or Launchpad shortcut could be added to the Dock.

2.2.2 Option 2

If pip cannot download ERRERS at step 2 above, you may be able to download the package manually from https://pypi.org/project/errers/#files.[5] If so:

1. Download the installation file `errers-VERSION-py3-none-any.whl`, where VERSION represents the version number of ERRERS that you wish to install.
2. Open a command prompt or shell from which Python can be run.
3. At the command prompt, change directory to the location of the installation file from step 1.
4. Type `python3 -m pip install WHEEL` and press enter, where WHEEL is the name of the installation file from step 1. On some systems, you may need to use `py` or `python` rather than `python3`. You may also need to use `pipx` rather than `pip`.
5. Continue with step 3 to 5 of option 1.

The installation procedures above fail if the *regex* module is not available. On Windows, they also fail if the *pywin32* package is not available. If for some reason these two packages cannot be installed, the installation can be forced by using option `--no-deps` when calling pip, which omits package dependencies: `python3 -m pip install --no-deps errers` or `python3 -m pip install --no-deps WHEEL`.[6]

2.3 Removal procedure

To remove ERRERS:

1. Open a command prompt or shell from which Python can be run. For instance, with the Anaconda Python distribution on Windows, you would choose Programs ⇒ Anaconda3 ⇒ Anaconda Prompt in the Windows Start menu.
2. Type `errers --shortcuts` and press enter.
3. In the shortcut-update window that appears, click *Delete* button to delete application shortcuts. If shortcuts had been copied to other locations, they must be deleted manually.
4. Type `python3 -m pip uninstall errers` and press enter. On some systems, you may need to use `py` or `python` rather than `python3`. You may also need to use `pipx` rather than `pip`.

[5] Installation files can also be downloaded from https://github.com/steve-guillouzic-gc/errers/releases, where pre-release versions may also be made available.

[6] Without the *regex* module, text extraction will be slower. Without the *pywin32* module on Windows, shortcut creation will be limited to the "Open With" menu, and the *Check* button—described in Section 3.1.1—will be missing.

3 Usage

ERRERS can be used in GUI or command-line interface (CLI) mode, respectively described in Sections 3.1 and 3.2. Section 3.3 lists some elements to keep in mind when using ERRERS.

3.1 Graphical user interface (GUI)

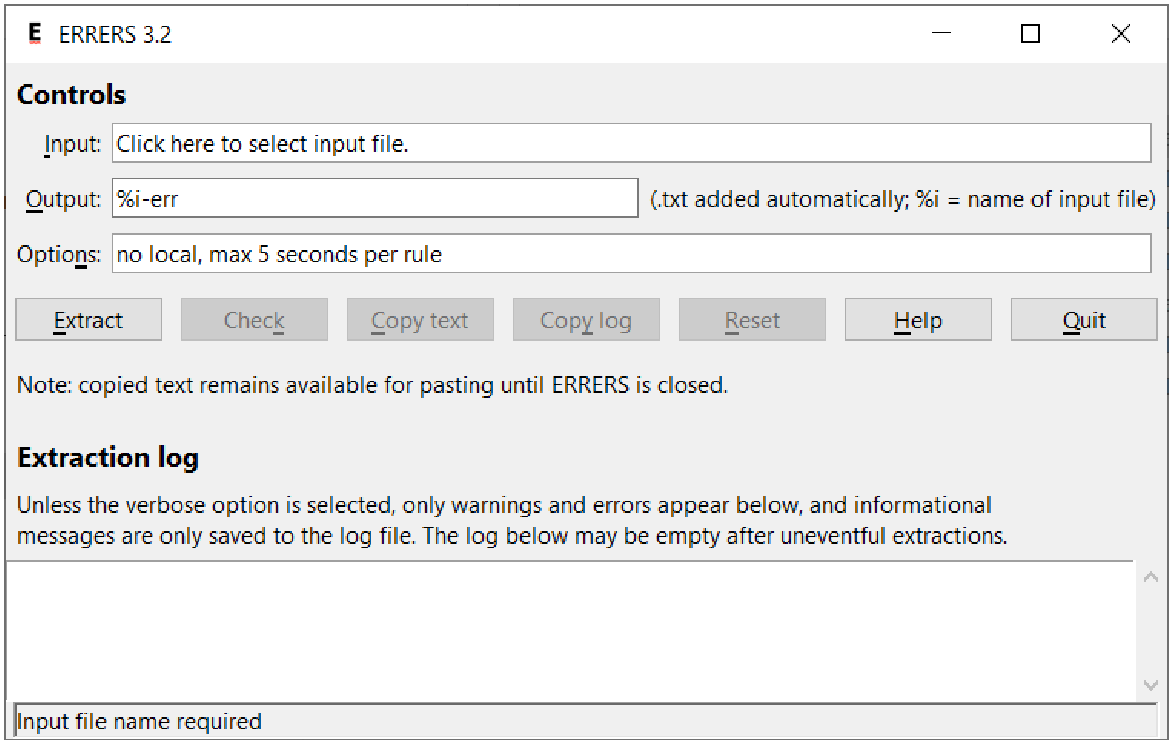
Screenshots of the ERRERS GUI on Windows are shown in Figure 2. The interface is similar on macOS and Linux, but there is no *Check* button on those platforms. The main window, in Figure 2a, is split into two sections: *Controls* and *Extraction log*. They are described in more detail in Sections 3.1.1 and 3.1.3, respectively. The options window, in Figure 2b, is described in Section 3.1.2.

3.1.1 Controls

The *Controls* section is where the input and output file names are specified and where buttons for running ERRERS are located. It also provides a summary of the options selected in the options window. 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, `%i` stands for the name stem of the input file (in other words, the file name without extension). The *.txt* extension 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 `%i-err`, which means that an input file name of `myreport.tex` would result in an output file name of `myreport-err.txt`. Clicking on the options field opens the options window. It is described in more detail in the next section.

The first button is called *Extract*. Clicking it launches the extraction process, with warnings and errors written to the *Extraction log* section. The entire log, including informational messages, is also saved to a file called `%o-log.txt`, where `%o` is the name stem of the output file. Once the extraction is done, the result can be copied to the clipboard by clicking on the *Copy text* button and pasted into a Microsoft Word document or another software for grammar and spell checking. (It is worth noting that this also works with the web-based version of Microsoft Word.) The log file can be similarly copied to the clipboard by clicking on the *Copy log* button to help report errors and missing substitution rules to the ERRERS developer. On Windows, the user can also click on the *Check* button to open the output file in Microsoft Word and initiate the spelling and grammar checking, after being given the opportunity to validate the language variants detected by Word. Clicking the *Reset* button prepares the GUI for re-running the extraction, whether with different runtime options or after amending the input file. (Changing the input file, the output pattern, or the option values also resets the GUI.) Finally, the *Help* button opens the help window, and the *Quit* button quits the application.

Keyboard shortcuts are provided for the various GUI elements. On Windows and Linux, the shortcut is the alt key combined with the letter underlined in the widget label. The only exception is the Times option, in Section 3.1.2, for which the shortcut is Alt+X. On macOS, the control key is used instead of alt. In dialog boxes, the enter and return keys can also be used for *Yes* and *Ok*, while the escape key can be used for *No* and *Cancel*. Finally, the tab key cycles through controls, and the space bar toggles checkboxes and activates buttons.



Controls

Input: Click here to select input file.

Output: %i-err (.txt added automatically; %i = name of input file)

Options: no local, max 5 seconds per rule

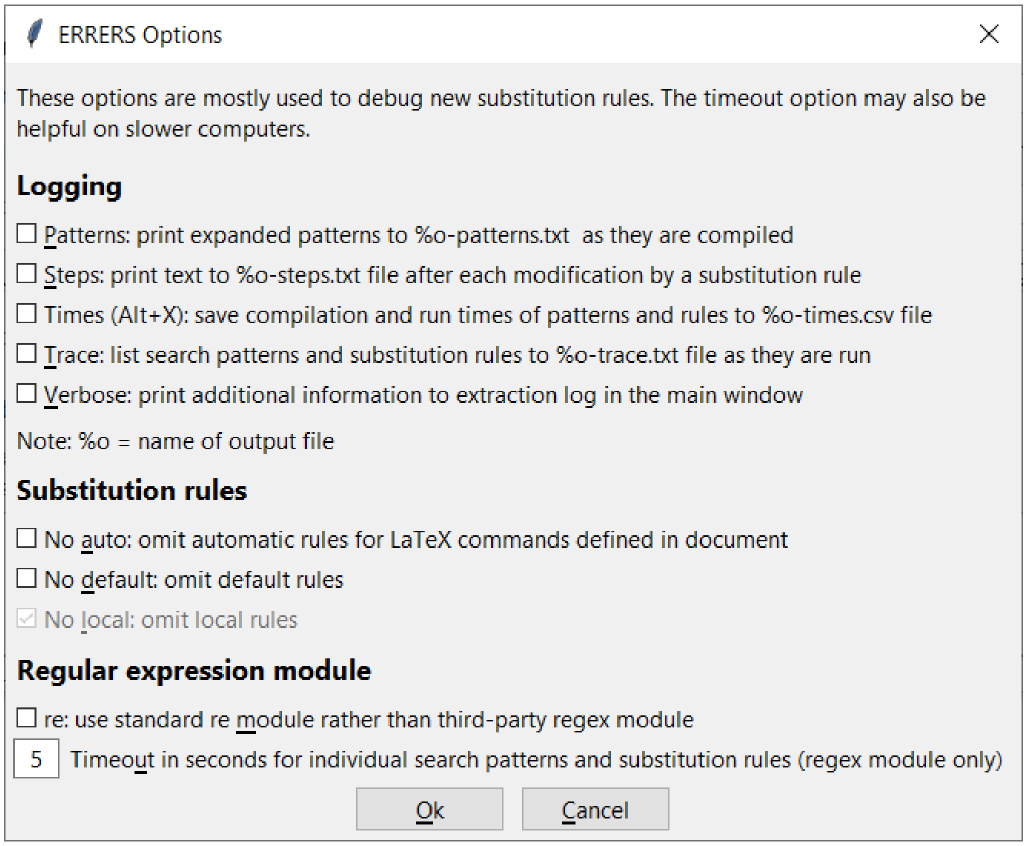
Note: copied text remains available for pasting until ERRERS is closed.

Extraction log

Unless the verbose option is selected, only warnings and errors appear below, and informational messages are only saved to the log file. The log below may be empty after uneventful extractions.

Input file name required

(a) Main window.



These options are mostly used to debug new substitution rules. The timeout option may also be helpful on slower computers.

Logging

☐ Patterns: print expanded patterns to %o-patterns.txt as they are compiled

☐ Steps: print text to %o-steps.txt file after each modification by a substitution rule

☐ Times (Alt+X): save compilation and run times of patterns and rules to %o-times.csv file

☐ Trace: list search patterns and substitution rules to %o-trace.txt file as they are run

☐ Verbose: print additional information to extraction log in the main window

Note: %o = name of output file

Substitution rules

☐ No auto: omit automatic rules for LaTeX commands defined in document

☐ No default: omit default rules

☑ No local: omit local rules

Regular expression module

☐ re: use standard re module rather than third-party regex module

5 Timeout in seconds for individual search patterns and substitution rules (regex module only)

(b) Options window.

Figure 2: *ERRERS GUI (Version 3.2) on Windows 10.*

3.1.2 Runtime options

The options window provides various options that can help with debugging when creating or updating substitution rules. They are split into three groups: logging, substitution rules, and regular expression module. The next three sub-sections provide a brief description of these options, and Section 4.3 goes into more detail.

3.1.2.1 Logging

These options activate the logging of additional information (`%o` stands for the name stem of the output file):

1. **patterns:** Print the expanded form of search patterns to a file named `%o-patterns.txt` as they are compiled;
2. **steps:** Print the extracted text (in its current state) to a file named `%o-steps.txt` after each matching substitution rule;
3. **times:** Save the compilation and run times for the search patterns and substitution rules to a comma-separated values (CSV) file named `%o-times.csv`;
4. **trace:** Print the search patterns and substitution rules to a file named `%o-trace.txt` as they are run; and
5. **verbose:** Print informational messages to the *Extraction log* box in addition to warnings and errors (the entire log is always saved to `%o-log.txt` whether this option is selected or not).

3.1.2.2 Substitution rules

These options deactivate certain sets of rules:

1. **no auto:** Omit automatic rules for LaTeX commands defined in document;
2. **no default:** Omit default rules that ERRERS uses for LaTeX commands that have no explicit rules defined;[7] and
3. **no local:** Omit local rules defined in the *errers.rules.local* module (see Annex A).[8]

3.1.2.3 Regular expression module

These options relate directly to the regular expression module:

1. **re:** Use the *re* module even if the *regex* module is available; and
2. **timeout:** Timeout in seconds for individual search patterns and substitution rules (with *regex* module only).

[7] ERRERS 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.
[8] When there is no such module, the option is checked and greyed out.

3.1.3 Extraction log

ERRERS writes warnings and errors to this section, such as:

1. Warnings about how using the *re* module deactivates the automatic detection of catastrophic backtracking while increasing the risk of encountering it;
2. List of LaTeX commands left at the end of the extraction, if any;
3. Any syntax error encountered in regular expressions; and
4. Any Python error that occurs during program execution.

When the verbose option is selected, it also writes the following information:

1. Path of input and output folders;
2. ERRERS version and whether it was packaged as a bundled application;
3. Version of Python interpreter;
4. Name and version of regular expression module (and version of *pywin32* package if applicable[9]);
5. Operating system and processor class;
6. Whether automatic, default, and local rules were applied;
7. Whether a LaTeX log file was found to help identify document class and packages;
8. Name of all files loaded during the extraction;
9. Rules defined in the LaTeX document; and
10. Rule functions from which rules were obtained.

When using the *re* module with the *trace* option, search patterns and substitution rules are also written to the extraction log as they are executed, to help diagnose catastrophic backtracking. (As mentioned above, automatic detection of catastrophic backtracking is not available with the *re* module.)

3.2 Command-line interface (CLI)

ERRERS can be run by typing `errers` followed by the desired command-line arguments. (See Figure 3.) If an input file argument is not specified, ERRERS starts in GUI mode. If the `-h` option is specified, a help message is printed. The `--gui` option forces ERRERS to start in GUI mode even when an input file is specified, the `--version` option prints out the version number of ERRERS, and the `--shortcuts` option opens the shortcut-update window used for installation and removal. Other arguments correspond to their GUI equivalents.

3.3 Usage notes and limitations

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

[9] The *pywin32* package provides a Python interface to Microsoft Windows libraries [11].

```
usage: errers [--gui] [--help] [--outfile OUTFILE] [--shortcuts] [--version]
              [--patterns] [--steps] [--times] [--trace] [--verbose]
              [--no-auto] [--no-default] [--no-local] [--re]
              [--timeout SECONDS]
              [INFILE.tex]

Positional argument:
  INFILE.tex    input file

General options:
  --gui         launch in GUI mode
  --help, -h    show this help message and exit
  --outfile OUTFILE, -o OUTFILE
                pattern for name stem of output file (name without extension),
                with %i standing for name stem of input file; .txt extension
                added automatically; default: %i-err; also determines names of
                log, pattern, step, time, and trace files: OUTFILE-log.txt,
                OUTFILE-patterns.txt, OUTFILE-steps.txt, OUTFILE-times.txt, and
                OUTFILE-trace.txt
  --shortcuts   launch shortcut-update GUI
  --version     print out version number and exit

Debugging options (logging):
  --patterns    print expanded patterns to OUTFILE-patterns.txt as they are
                compiled, to verify that expansions work as expected
  --steps       print text to OUTFILE-steps.txt after each matching rule, to
                help debug interactions between them
  --times       save compilation and run times of regular expressions to
                OUTFILE-times.csv
  --trace       list patterns and rules to OUTFILE-trace.txt as they are run,
                to help identify source of catastrophic backtracking
  --verbose     print informational messages to standard error in addition to
                warnings and errors; also stream the trace if --trace is
                specified and the standard re module is used (because automatic
                detection of catastrophic backtracking is not available with
                that module)

Debugging options (rule selection):
  --no-auto     omit automatic substitution rules for LaTeX commands defined in
                document
  --no-default  omit default substitution rules, to help debug command-specific
                rules
  --no-local    omit local substitution rules

Debugging options (regular expression module):
  --re          use standard re module even if third-party regex module is
                available
  --timeout SECONDS
                timeout in seconds for individual search patterns and
                substitution rules used as indication of likely catastrophic
                backtracking when using the regex module; default: 5 seconds
```

Figure 3: *Command-line arguments and options of ERRERS (Version 3.2).*

3.3.1 LaTeX log file or `\usepackage` commands

LaTeX packages often depend on other packages and load them on 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, ERRERS uses the log file produced by LaTeX to identify those classes and packages. However, if the LaTeX log file is not available, ERRERS only loads rules for the document class specified using **`\documentclass`** and for packages loaded using **`\usepackage`**.

For instance, the Ti*k*Z 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, ERRERS 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 ERRERS rule for `\definecolor` is only applied if **`\usepackage{xcolor}`** appears explicitly in the document. For ERRERS to work properly, it is thus important to process the LaTeX file before running ERRERS or to include an explicit **`\usepackage`** command for all packages that provide commands used in the document. Notwithstanding this, it is always a good idea to run LaTeX before ERRERS to ensure that there are no syntax errors in the document.

3.3.2 Default rule confusion

As mentioned in footnote 7 on page 7 and in Section 4.3.2, ERRERS 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 few different 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 to reduce the amount of document customization required, ERRERS provides explicit rules for several common one-argument commands such as **`\mbox`**, **`\emph`**, and **`\textbf`**.

3.3.3 Differences between *regex* and *re* modules

The *regex* module has several capabilities that are not present in the *re* module or were implemented recently. The following ones are used by ERRERS:

1. Recursive patterns, which reduce the number of times that rules need to be applied (see Annex B);
2. Atomic groups and possessive quantifiers, which reduce the risk of catastrophic backtracking;[10] and

[10] Atomic groups and possessive quantifiers were added to the *re* module in Python 3.11, but an implementation bug prevented their usage with ERRERS prior to Python 3.11.5.

3. Timeout, which is used to automate the detection of catastrophic backtracking.

Because of the absence of recursive patterns in 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`**, **`\newenvironment`**, **`\def`**, or similar commands. This issue does not arise with the *regex* module. If desired, the problem with the *re* module can be mitigated by placing the command definition between **`\makeatletter`** and **`\makeatother`**, as ERRERS discards all text between those two commands, and defining a substitution rule manually (see Section 4).

As mentioned above, the absence of recursive patterns in the *re* module means that more rules need to be applied iteratively, which increases the extraction time. Furthermore, ill-designed search patterns carry a higher risk of catastrophic backtracking with the *re* module prior to Python 3.11.5 because of the absence of atomic groups and possessive quantifiers. The problem is compounded by the absence of timeout, which precludes their automatic detection. Still, when catastrophic backtracking is encountered with the *re* module, the GUI freezes—particularly the extraction timer in the status bar. When that happens, the problematic substitution rule can be identified using the *trace* option. If the issue arises because of a rule created automatically for a command defined in the document using **`\newcommand`**, **`\newenvironment`**, **`\def`**, or similar commands, it can be mitigated by placing the command definition between a pair of **`\makeatletter`** and **`\makeatother`** and defining a substitution rule manually (see Section 4).

4 Substitution rules

As mentioned in the introduction, the text from LaTeX documents is extracted through the application of text substitution rules. The set of rules currently implemented in ERRERS covers LaTeX commands used over the years by the author and users of the software. Also, since version 3.1, ERRERS automatically creates rules for commands defined in a document using the `\newcommand`, `\newenvironment`, and `\def` families of commands, which significantly reduces the need for users to create their own substitution rules.

When needed, additional 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;[11]
2. **Local-rules module:** Rules for LaTeX commands that a user defines locally but uses in multiple documents may optionally be stored in a user-defined *errers.rules.local* module saved in the ERRERS installation folder (see Annex A); and
3. **Standard-rules module:** Rules for LaTeX commands that come from standard LaTeX packages should be sent to the ERRERS developer for inclusion into the standard rule repository (*errers.rules.standard*).

The extraction is performed in several phases. In each phase, rules defined manually in a LaTeX document are applied before those defined in the *errers.rules.local* and *errers.rules.standard* modules. Similarly, those in the *errers.rules.local* module are applied before those defined in the *errers.rules.standard* module. Rules created automatically for commands defined in the document are applied together with those defined in the *errers.rules.standard* module. (More information about the order of rule application is provided in Annex B.) This allows users to override rules provided with or created automatically by ERRERS, if desired. It also lets them write rules in terms of those provided with ERRERS, to avoid reinventing or duplicating complex rules and to ensure consistency if standard ERRERS rules are modified in future versions.[12]

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

4.1 Syntax

Substitution rules specified in LaTeX documents are put in comments to avoid interfering with compilation and must appear in the main LaTeX file rather than a sub-file. Each rule is written as "`% Rule(PATTERN, REPLACEMENT, iterative=ITERATIVE, phase=PHASE)`," where `%` must be the first character of the line, `PATTERN` is the search pattern, `REPLACEMENT` is the replacement specification, `ITERATIVE` is equal to `True` or `False`, and `PHASE` specifies the extraction phase at which the rule is applied. The `iterative` and `phase` arguments are optional. The four arguments are described in Sections 4.1.1 to 4.1.4. Each rule can be specified over multiple lines if required; in that case, each continuation line must also start with `%`. Putting a second percent sign at the beginning of the line prevents the rule from being detected and can be used to comment it out.

[11] When a LaTeX document is split into multiple files, document-specific substitution rules for ERRERS must be specified in the main file.

[12] An example of this is provided in Section 4.2.5. It shows how the rule for the `\subcaption` command is expressed in terms of the `\caption` command.

4.1.1 Search pattern

The search pattern is a regular expression. It uses the syntax defined in the documentation of the *re* module [9].[13] It supports possessive quantifiers and atomic groups, as they reduce the risk of catastrophic backtracking.[14] When the *re* module is used with a version of Python prior to 3.11.5, they are automatically replaced with greedy quantifiers and non-capturing groups, respectively.

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

1. The `%c`, `%r`, and `%s` strings 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 `%C` string is replaced with a regular expression pattern that matches a pair of curly braces with arbitrary content in between, the name of a LaTeX command, or a single character (which matches how LaTeX processes command arguments);
3. The `%m` string is replaced with a regular expression pattern that matches the names of LaTeX commands (such as `\textbf` or `\^`); and
4. The `%h`, `%n`, and `%w` strings 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`. Since `%C` can match arbitrary characters, it does not make sense to declare it as optional.

The content matched by `%C`, `%c`, `%r`, and `%s` is captured as named references:[15]

1. The capture group of the first `%c` or `%C` is called `c1`;
2. The capture group of the second `%c` or `%C` is called `c2`;
3. Etc.

For `%r`, capture groups are called `r1`, `r2`, etc. For `%s`, they are called `s1`, `s2`, etc. For placeholders that match bracketed content, the brackets are not included in the capture groups. The name of a capture group can be specified manually by appending an empty named group to its placeholder, such as in `%C(?P<custom_name>)`. An automatic name is not generated for such capture groups, and the index is not incremented. The text matched by `%m`, `%h`, `%n`, and `%w` is not captured.

[13] Regular expression syntax varies slightly between programming languages.

[14] 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.

[15] The patterns used for `%c`, `%r`, and `%s` include only one capture group each, but the one used for `%C` includes two (one for the bracket, if present, and one for the command argument). While this knowledge could allow someone to reference the capture groups by number rather than name, this is not supported and is discouraged, because future ERRERS releases may change the number of capture groups for these patterns.

4.1.2 Replacement specification

For every part of the text that matches the search pattern, ERRERS replaces the matched text with the one specified by the replacement string.[16] In the replacement string:

1. `\g<c1>` refers to the text matched by the first `%c` or `%C`;
2. `\g<c2>` refers to the text matched by the second `%c` or `%C`;
3. Etc.
4. `\g<r1>` refers to the text matched by the first `%r`;
5. `\g<r2>` refers to the text matched by the second `%r`;
6. Etc.
7. `\g<s1>` refers to the text matched by the first `%s`;
8. `\g<s2>` refers to the text matched by the second `%s`;
9. Etc.

Also, in the replacement string, `\n` refers to a newline character.

4.1.3 Iterative

The optional `iterative` argument indicates if the rule should be applied iteratively. The default value is `False`. Each rule for which `iterative=True` is applied repeatedly until it no longer matches the text.

4.1.4 Phase

The optional `phase` argument indicates when the rule is applied. It can be set to `'location'`, `'insertion'`, `'removal'`, `'setup'`, `'main'`, or `'cleanup'`. The default value is `'main'`. These phases are described in Annex A.

4.2 Examples

The examples provided in this section are extracted from the *errers.rules.standard* module, 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 backslashes as special characters. This is not needed for the replacement specification, because it does not contain

[16] For rules defined in LaTeX documents, replacement specifications must be strings. For rules defined in the *errers.rules.local* and *errers.rules.standard* modules, they can also be functions or classes. (See Annex A.)

any backslash character. The backslash is also a special character in regular expressions. To represent a single backslash in LaTeX, a double backslash must therefore be written in the regular expression. This is why the search pattern starts with a double backslash.

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\g<s1>\n\n\g<c1>\n'),
```

In this rule, the search pattern is written as a multi-line string using triple quotes. Since regular expressions in ERRERS are interpreted using the verbose option [9], 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 `%c` rather than `%C` placeholder is used for the mandatory argument because, when using the *re* module, `%C` could match the opening square bracket of an optional argument if the actual mandatory argument did not match due to having more than two levels of curly braces.

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. If 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[17]

```
% Rule(r'(?s)\\footnote(?:text)?%s?%c(?P<rest_of_para>.*?)\n%h\n',
%       r'\g<rest_of_para> (\g<c1>)\n\n', iterative=True),
```

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.[18] 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, `(?P<rest_of_para>.*?)\n%h\n` matches the rest

[17] The rule in the *errers.rules.standard* module differs slightly from the one presented here. It uses a function as replacement specification rather than a string (see Annex A), which allows it to remove white space around the footnote argument.

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

of the paragraph after the end of the footnote up to the next empty line and assigns it to the capture group named `rest_of_para`.

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=True` argument makes ERRERS apply the rule 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[\g<s1>]{\g<c1>}')
```

In this rule, the **`\subcaption`** command from the subcaption package is replaced by the core **`\caption`** command, the rule for which is applied later in the extraction process. 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 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 Runtime options

As mentioned in Section 3.1.2, ERRERS provides several options to help debug rules when they do not produce the anticipated result. They are described in more detail in this section, where they are listed in alphabetical order. Annex C provides additional debugging information for advanced users.

4.3.1 No auto

ERRERS creates rules automatically for commands defined in the LaTeX document using **`\newcommand`**, **`\renewcommand`**, **`\providecommand`**, **`\newenvironment`**, **`\renewenvironment`**, **`\def`**, **`\edef`**, **`\gdef`**, and **`\xdef`**. It also creates a **`\the`**… command for each counter defined using **`\newcounter`**. The *no auto* option prevents the creation of such rules to help assess, when needed, if extraction problems could be caused by one of them. The creation of automatic rules can be deactivated for specific command definitions by placing them between **`\makeatletter`** and **`\makeatother`** pairs. ERRERS discards everything placed between these commands.

4.3.2 No default

ERRERS 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 *no default* option deactivates default rules so users can see more clearly how well the command-specific rules work.

4.3.3 No local

ERRERS allows the user to create a repository of local rules in the *errers.rules.local* module (see Annex A). The *no local* option deactivates these rules to help diagnose if a problem is caused by local rules.

4.3.4 Patterns

When a search pattern does not match text as expected, it may be useful to look at the exact search pattern generated by ERRERS to see more precisely what is happening. When the *patterns* option is specified, ERRERS prints the generated search patterns to a `%o-patterns.txt` file as they are compiled.

4.3.5 *re* module

By default, ERRERS uses the *regex* module whenever available because it leads to a faster and more robust execution. The *re* option forces it to use the *re* module even if the *regex* module is available. This option is used to verify that the two modules yield the same output.

4.3.6 Steps

Sometimes, problems with the text extraction 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 to analyze those interactions. When the *steps* option is specified, ERRERS 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 extraction. 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 13 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;
- **Line 7:** rule definition;
- **Line 8:** number of substitutions performed;
- **Line 9:** separator between the description of the rule and the output resulting from its application; and
- **Lines 10 to 13:** 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 15 and 23) were default rules that removed all remaining `\begin` and `\end` commands (there was only one of each); and
- The fourth rule (line 30) removed the remaining empty line at the beginning of the document.

```
\documentclass{article}
\begin{document}
Hello world!
\end{document}
================================================================================
standard.py, line 551, core_main:
Rule(r'\\documentclass%s?%c', '')
Substitutions: 1
--------------------------------------------------------------------------------

\begin{document}
Hello world!
\end{document}
================================================================================
standard.py, line 706, core_cleanup:
Rule(r'\\begin%C(?:%c|%r|%s)*+%n', '')
Substitutions: 1
--------------------------------------------------------------------------------

Hello world!
\end{document}
================================================================================
standard.py, line 707, core_cleanup:
Rule(r'\\end%C%n', '')
Substitutions: 1
--------------------------------------------------------------------------------

Hello world!
================================================================================
standard.py, line 732, core_cleanup:
Rule(r'\A\n++', '')
Substitutions: 1
--------------------------------------------------------------------------------
Hello world!
================================================================================
```

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

4.3.7 Timeout

The *regex* module provides the option of specifying a timeout value when applying regular expressions. ERRERS leverages this to detect catastrophic backtracking. By default, ERRERS uses a timeout value of 5 seconds per regular-expression operation. If an operation times out, ERRERS reports which search pattern or substitution rule triggered it, so the user can review the pattern or rule for sources of catastrophic backtracking. Operations may sometimes timeout erroneously if the computer is running other intensive software at the same time or if ERRERS is applied to a huge LaTeX document. In those cases, it may be helpful to increase the timeout value.

The *re* module does not support such timeout. As mentioned in Section 3.3.3, the timer in the status bar of the GUI freezes when encountering catastrophic backtracking with the *re* module. When that happens, the *trace* option can be used to determine which pattern or rule is responsible.

4.3.8 Times

When the *times* option is selected, ERRERS saves the compilation and run times of the search patterns and substitution rules to a CSV file. This allows the user to see which rules are being applied and to assess their runtime efficiency. The CSV file contains the following columns:

1. **File:** name of file where rule is defined (one of the ERRERS modules or the LaTeX document);
2. **Line:** line number 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 substitution rule was run;
7. **Matches:** number of times that search pattern or substitution rule matched the text, summed over all runs; and
8. **Object:** definition of search pattern or substitution rule.

On Windows, the compilation and run times correspond to clock time; on macOS and Linux, they correspond to central processing unit (CPU) time.

4.3.9 Trace

When the *trace* option is selected, ERRERS prints each search pattern and substitution rule to the debugging log before running it. This allows the identification of the culprit when a search pattern or substitution rule faces catastrophic backtracking. Lines of the trace log are similar to lines 6–7, 15–16, 23–24, and 30–31 in 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 substitution rule is applied, even if it does not match. For the example shown in Figure 4, the trace contains 3139 lines.

4.3.10 Verbose

ERRERS always prints warnings and errors to the *Extraction log* when running the GUI or to standard error otherwise. When the *verbose* option is selected, additional diagnostic information is also printed, as indicated in Section 3.1.3. This information is always written to the `%o-log.txt` file, whether or not the *verbose* option is selected.

5 Conclusion

ERRERS extracts the text from LaTeX documents, so their grammar and spelling can be checked using tools such as Microsoft Word. It is based on text substitution rules defined using regular expressions. Rules are provided for many LaTeX commands already, and additional rules are created automatically for commands defined in the document. Additional rules can be defined as needed by users on a per-document basis for commands that are not yet supported by ERRERS or to override the rules provided with ERRERS to customize the output. While not mandatory, users can create a local rule repository for custom commands used in multiple documents. If users create rules for additional commands from the standard LaTeX classes or packages or from the standard BibTeX styles, they should forward them to the ERRERS developer for inclusion into the base product, so that others can benefit from them. ERRERS 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] Office VBA Reference (online), Microsoft, https://learn.microsoft.com/en-us/office/vba (Access Date: March 2023).

[5] Learn Vimscript the Hard Way (online), Steve Losh, https://learnvimscriptthehardway.stevelosh.com/ (Access Date: March 2023).

[6] Python (online), Python Software Foundation, https://www.python.org/ (Access Date: March 2023).

[7] GNU Operating System (online), Free Software Foundation, https://www.gnu.org/ (Access Date: March 2023).

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

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

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

[11] pywin32 (online), Mark Hammond et al., https://pypi.org/project/pywin32/ (Access Date: March 2023).

[12] Installing Packages (online), Python Software Foundation, https://packaging.python.org/en/latest/tutorials/installing-packages/ (Access Date: March 2023).

Annex A Local rules

It is generally sufficient to keep custom substitution rules in the LaTeX document itself. However, if a user wants to use the same custom rules for multiple documents, they can be placed in an *errers.rules.local* module saved in the ERRERS installation folder. This is done by putting a `local.py` file in the same folder as the `standard.py` file that provides the *errers.rules.standard* module. Figure A.1 provides a fictitious example of such a file.

```
from errers.rules import standard

def core_main(Rule, **_):
    return Rule(r'\\ref%C', '42')

def class_drdc_report_main(Rule, **_):
    return Rule(r'\\projectnumber%1', '')

def package_onetwo_main(Rule, RuleList, **_):
    return RuleList([
        Rule(r'\\one%c%c', r'\1'),
        Rule(r'\\two%c%c', r'\2')
    ])

style_drdc_custom_main = standard.style_drdc_main
```

Figure A.1: *Fictitious `local.py` file.*

The `local.py` and `standard.py` files define various functions that return individual substitution rules or lists of rules. These functions are called rule functions, and their names determine when the rules are applied. The name is composed of two parts:

1. The beginning of the function name determines if the rules are applied for a given document; and
2. The end of the function name determines at which phase of the extraction the rules are applied.

The first part of the name can take the following values:

1. **`core`:** rules applied for all documents;
2. **`class_X`:** rules applied for documents that load the document class X;
3. **`package_X`:** rules for documents that use package X; and
4. **`style_X`:** rules for documents that use the BibTeX style X.[19]

When class, package, or style names contain hyphens or periods, they are replaced with underscores in rule 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 onetwo package, and `rules_style_drdc_custom` returns rules for a custom version of the DRDC BibTeX style called drdc-custom.

The second part of the name can take the following values, listed in order of rule application:

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

1. **`location:`** rules applied once to main document at start of extraction, and applied individually to each file that is inserted into the main document, to record the file and line number of LaTeX command definitions;

2. **`insertion:`** rules applied once at start of extraction to insert other files into the main document;

3. **`removal:`** rules applied once at start of extraction to remove parts of the documents that do not generally need grammar and spell checking and can be removed, such as equations, verbatim environments, and comments;

4. **`setup:`** rules applied once at start of extraction because they do not need to be applied repeatedly even when using the *re* module;

5. **`main:`** rules that need to be applied repeatedly when using the *re* module, which are most of the rules;

6. **`cleanup_braces:`** default rules applied after each application of main rules to remove one level of single-argument commands and braces that do not wrap a command argument; and

7. **`cleanup:`** rules applied once at the end to the extraction to remove the **`\begin`** and **`\end`** commands, the remaining no-argument commands, and superfluous spaces and newline characters.

These phases correspond to those listed in Section 4.1.4, except `'cleanup_braces'`, which is only available in the *errers.rules.standard* and *errers.rules.local* modules and only for core rule functions. If a rule function with a given name is present in both *errers.rules.standard* and *errers.rules.local*, the rules returned by the function in *errers.rules.standard* are applied immediately after those returned by the one in *errers.rules.local*.

Rule functions have access to the following ten keyword arguments:

1. `Rule`: class used to create extraction rules;

2. `RuleList`: class used to group extraction rules;

3. `logger`: logging object for informational messages, warnings, and errors;

4. `auto`: Boolean indicating whether to create rules dynamically for commands defined in LaTeX document using **`\newcommand`**, **`\newenvironment`**, **`\def`**, and similar commands;

5. `default`: Boolean indicating whether to apply default rules for LaTeX commands that take one or no argument, and for unknown LaTeX environments;

6. `document`: object providing read-only interface to main LaTeX file;

7. `read_file`: function that inserts a file into the document (not available for `'location'` rules);

8. `not_commented`: pattern prefix ensuring that LaTeX command is not in a comment (useful only up to removal phase, where comments are removed);

9. `not_escaped`: pattern prefix ensuring that following character is not escaped while recognizing newline commands (useful only up to setup phase, where newline commands are removed; a simpler pattern can be used in later phases); and

10. `single_pass`: Boolean indicating that the engine is going through the rules only once, which is indicative of using the *regex* module rather than the built-in *re* module.

Most rule functions use only the first two arguments. Those that need to log messages also use `logger`. The other ones are more specialized and are unlikely to be needed for most rule functions. Rule functions typically use the underscore variable name to capture unused keyword arguments.

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

1. **Replace existing rules:** the rules returned by the `core_main` and `class_drdc_report` functions in the *errers.rules.local* module are executed before those returned by the functions of the same names in the *errers.rules.standard* module, which means that the standard rules for `\ref` and `\projectnumber` are overshadowed by the local ones;
2. **Define rules for additional commands:** the `rules_package_onetwo` function defines rules for two commands of the fictitious onetwo package; and
3. **Indicate that a rule list defined for one class, package, or style should be used for another:** defining `style_drdc_custom_main` as an alias for the `style_drdc_main` function defined in the *errers.rules.standard* module means that the main phase rules for the standard DRDC BibTeX style are also applied to documents that use the locally defined drdc-custom style.

For rules defined in LaTeX documents, the replacement text is always specified using a string. There is however more flexibility for rules defined in the *errers.rules.standard* and *errers.rules.local* modules. In both files, 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 [9]; or
3. **Class of function object:** similar to using a function, but the replacement text can vary based on the matching order.[20]

[20] Such a function object is used for instance by the rules that replace `%C`, `%c`, `%r`, and `%s` with search patterns for command arguments, to iterate through the named groups (c1, c2, etc.).

Annex B Order of rule application

ERRERS applies substitution rules in phases, in the order listed in Section A:

1. `location`;
2. `insertion`;
3. `removal`;
4. `setup`;
5. `main`;
6. `cleanup_braces`; and
7. `cleanup`.

In each phase, rules are applied in the following order:

1. Rules defined in LaTeX document;
2. Rules for document classes, packages, and bibliography style; and
3. Core rules.

When rule functions with a given name are present in both *errers.rules.standard* and *errers.rules.local*, the local rules are applied before the standard ones. Rules applied earlier in the extraction process can overrule latter rules or be defined in terms of them.

When using the *regex* module, ERRERS only needs to apply each phase once in the order above. However, rules from the `location` phase are also applied to each file inserted into the main document.[21] When the *re* module is used, rules from the `main` phase are applied iteratively until none of them match; if any rule from the `cleanup_braces` phase matches, the process goes back to the `main` phase.

The difference between the two modules comes from the fact that the *regex* module supports recursive patterns but the *re* module does not. The recursive patterns allow ERRERS to identify command arguments irrespective of the number of balanced bracket levels in each of them.[22] Without recursive patterns, the substitution rules are limited in the number of balanced bracket levels that are allowed in arguments: they only allow one level in addition to the one surrounding the argument.[23] 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, ERRERS with the *re* module must process the commands from the inside out and needs to apply the rules multiple times to process all of them.

[21] Furthermore, some individual rules may be declared as iterative.

[22] 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.

[23] 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, ERRERS 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 ERRERS for the **`\eqref`** command of the amsmath package. It does the same thing as the core **`\ref`** command, but in parentheses 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{\g<c1>})')
2  \eqref{eq}
```

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

If the substitution rule referenced the non-existing capturing group c2 in the replacement specification, the regular expression engine would raise an invalid capturing group error. ERRERS 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.[24] The second line of the report contains the faulty rule definition.

```
1  Error in replacement string (example.tex, line 1, ): unknown group
2  Rule(r'\\eqref%C', r'(\\ref{\g<c2>})')
3  Extraction interrupted by regular expression error.
```

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 ERRERS 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 463 (indexed from zero). Lines 2 to 14 contain the regular expression up to the error point. Line 15 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 16 to 51. (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 ERRERS uses the engine's verbose option.)

The search pattern displayed by ERRERS 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 pattern as defined in ERRERS or by the user, whereas Figure C.3 shows it as seen by the regular expression engine to show the location of the error. As mentioned in Section 4.1.1, ERRERS replaces the `%C` appearing in the search pattern of the **`\eqref`** rule by a regular expression that matches a LaTeX argument in curly brackets, the name of LaTeX command, or a single character—as shown in lines 17 to 51 of Figure C.3. The version shown here is for the *regex* module; the one for the *re* module is slightly different. ERRERS also prepends the search pattern with three negative lookbehind patterns. The first one ensures that the rule matches

[24] If the rule were defined in *errers.rules.standard* or *errers.rules.local*, the name of the class or function where the definition is located would be listed after the line number.

```
Error in search pattern (example.tex, line 1, ): unbalanced parenthesis at position
463 (line 13, column 5)

                   # NEGATIVE LOOK-BEHIND PATTERNS
(?<!               # FIRST:
    (?<!\\)        # Can follow pair of backslashes,
    \\             # but not a unique one.
)                  #
(?<!               # SECOND:
    \\newcommand{  # Must not be first argument
)                  # of command definition (\newcommand).
(?<!               # THIRD:
    \\def{         # Must not be first argument
)                  # of command definition (\def).
\\eq
----|
    )ref
                                              # CURLY-BRACKET ARGUMENT
(?>                                           # Atomic non-capt group for quantifiers
    [\ \t]*+\n?+[\ \t]*+(?:%.*+\n[\ \t]*+)*+  # Drop white space (one \n max)
    (?P<c1_ob>(?<!\\){)?+                     # Opening bracket (optional)
        (?P<c1>                               # Start capturing group
            (?<=(?<!\\){)                     # Case 1: bracketed content
            (?:                               # Non-capt group for alternative
                (?>(?!(?<!\\){)(?!(?<!\\)})(?s:.))++
                                              # Non-brackets
                |                             # Or
                (?:                           # Balanced brackets
                    (?<!\\){                  # Opening bracket
                        (?&c1)                # Recursive pattern
                    (?<!\\)}                  # Closing bracket
                )
            )*+                               # Capture as much as possible
            |                                 # Or
            (?<!(?<!\\){)                     # Case 2: non-bracketed LaTeX macro

                                  # COMMAND NAME
                \\                # Initial backslash
                (?:               # Non-capturing group for alternative
                    [a-zA-Z]++    # Letters
                    |             # Or
                    \s            # Single space
                    |             # Or
                    .             # Any other character
                )                 # End non-capt group

            |                                 # Or
            (?<!(?<!\\){)                     # Case 3: non-bracketed character
            (?![\ \t\n])(?>(?!(?<!\\){)(?!(?<!\\)})(?s:.))
        )                                     # End capturing group
    (?(c1_ob)(?<!\\)})                        # Closing bracket (case 1 only)
)                                             # End non-capt group

Extraction interrupted by regular expression error.
```

Figure C.3: *Syntax error if unbalanced parenthesis is present.*

`\eqref` but not `\\eqref`.[25] The other two ensure that `\eqref` is not the first argument of a command definition. While not applicable here, when a search pattern ends with an argument-less LaTeX command, ERRERS appends negative lookahead patterns 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`. The second of those lookahead patterns is also appended to the command name in rules for which the first argument is matched by `\%C`, since this pattern can match arbitrary individual characters.

[25] 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
ERRERS	Enhanced Review of Reports via Extraction using Rule-based Substitutions
GNU	GNU's Not Unix
GUI	graphical user interface
VBA	Visual Basic for Applications

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.)

ERRERS 3.2 (Enhanced Review of Reports via Extraction using Rule-based Substitutions): 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	38	12

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; spelling; user manual; regular expression; Python

13a. ABSTRACT (When available in the document, the English version of the abstract must be included here.)

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

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

ERRERS est un logiciel programmé en langage Python qui extrait le texte de documents LaTeX de façon à ce que l'on puisse vérifier la grammaire et l'orthographe avec des outils comme Microsoft Word. Il est fondé sur des règles de substitution de texte définies à l'aide d'expressions régulières. Des règles sont fournies pour de nombreuses commandes LaTeX, et des règles supplémentaires sont créées automatiquement pour les commandes définies dans des documents. Les utilisateurs peuvent définir des règles additionnelles pour traiter les commandes qui ne sont pas encore prises en charge par ERRERS ou qui sont définies localement. Ils peuvent aussi remplacer les règles fournies avec ERRERS 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 substitution de texte.