LaTeXBuddy

Release 0.5.0

LaTeXBuddy authors

Apr 02, 2023

ABOUT

The only LaTeX checking tool you'll ever need.

LaTeXBuddy is the checking tool for LaTeX, which combines the power of various other tools in one easy-to-use command-line tool with clear HTML output. Aspell, ChkTeX, LanguageTool: you name it! LaTeXBuddy is modular and Python-based, so that implementing new functionality becomes a breeze!

Important: LaTeXBuddy is a **work in progress**. We are working on fixing bugs and cleaning up. Using LaTeXBuddy in its current state may come with a lot of inconveniences. Upon reaching the Beta status, we will open-source this project. For now, it technically remains copyrighted, yet you're free to fork it and provide your edits and improvements to the code base.

CHAPTER

ONE

CHANGELOG

All notable changes to LaTeXBuddy will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

1.1 Unreleased

Warning: These changes have **not** been part of a release yet.

No significant changes.

1.2 0.5.0 - 2023-04-02

1.2.1 BREAKING CHANGES

- #112: CLI got revamped and simplified
  - **removed** option `--flask` use executable `latexbuddy-server` instead
  - **removed** options `--wl_add_keys` and `--wl_from_wordlist` use executable `latexbuddy-whitelist` instead
  - whitelist operations got moved to the new `whitelist` module
  - removed the big mutually exclusive group, which should affect usage
- `extend_path()` call was removed, making LaTeXBuddy not a namespace any more

1.2.2 Changed

- #105: LaTeXBuddy is now open-source under the terms of GPL-3.0-or-later
- #110: some calls to `os.path` were replaced with `pathlib.Path`
- Replaced usage of MD5 with SHA-1 and marked this usage as insecure. This should allow execution of LaTeXBuddy in protected environments and on FIPS builds of Python.
- Requests to the LanguageTool API now have a timeout of 60 seconds.

1.2.3 Fixed

- fix a couple documentation issues, like missing modules and broken links

1.2.4 Behind-the-scenes

- #125: Poetry was ditched in favour of a rather vanilla setuptools+tox setup
- #126: Enabled test results output in GitLab CI.
- #127: Automated release publishing. Now, we can publish a new Git tag, and this will automatically create a GitLab release
- Enabled code coverage calculation on each test.
- Improved CI
  - smoke test is now being run in parallel on multiple Python versions
  - improved caching of pip between jobs and branches
- Not adding changelog entries to a merge request will now raise a warning in CI.
- Towncrier is now used to generate the changelog

1.3 0.4.2 - 25 Dec 2022 :christmas_tree:

1.3.1 Fixed

- fixed regression introduced in ef2e4e2f9ff2ac3e1a9772cfec0985b6b4e20d9c where the app would crash because of a weird typing error (!180)

1.3.2 Changed

- `TexFile` will not try reading a file if it's empty, but return an empty string (#22, !177)
- Logging was simplified (!181)
- replaced custom `get_app_dir()` with a more robust `platformdirs` (!182)

1.4 0.4.1 - 09 Dec 2022

1.4.1 Fixed

- the latest version got published with the wrong tag (0.3.0)

1.5 0.4.0 - 09 Dec 2022

1.5.1 BREAKING CHANGES

- minimal Python version set to 3.7 (!168)

1.5.2 Added

- line numbers (!135)
- new test cases for multiple occurrences of own_checkers problems (!110)
- custom key for YaLafi problems (!109)
- filter for log files (!121)
- new base class for all modules and LatexBuddy (`NamedModule`) (!108)
- new `Loggable` base class which provides a properly named logger to any class inheriting from it (!108)
- new module "NewerPublications" that checks for each entry in the BibTeX file if a newer publication exists (!120)
- new module "BibtexDuplicates" that checks the BibTeX file for similar entries (!120)
- debug message for beginning and end of whitelist check in `LatexBuddy` (!141)
- pytest environment (!142)
- all unit tests as per documentation (!142)
- all integration tests as per documentation (!142)
- default tooltip in html for problems without a custom description (!150)
- new test routines for HTML highlighter (!150)
- flask server as a GUI for checking documents (!154)

1.5.3 Changed

- the problem list and text is now scrollable (!135)
- language selection for aspell now works dynamically and using the config (!105)
- language codes are now standardized to fit different formats (!116)
- methods in ConfigLoader now take an instance or a type-descriptor of type NamedModule instead of taking the name as a string (!108)
- Problem API now takes an instance or a type-descriptor of type NamedModule instead of a string (!108)
- `NamedModule` is now the base class of `Module` and `MainModule` (therefore `LatexBuddy`) and provides a logger to all these classes by inheriting from `Loggable` (!108)
- all modules now use the new standards for ConfigLoader, Problem API and logging (!1ß8)
- `LatexBuddy` is now a singleton and inherits from `MainModule`, making it an instance of `NamedModule` as well (!108)
- modified format of `config.py`: options with key `"buddy"` are now located in a seperate dictionary (!108)
- languagetool now dynamically retrieves a list of supported languages from the commandline or (local/remote) server instead of comparing with a hardcoded list (!139)

- renamed tool_loader.py to module_loader.py and ToolLoader to ModuleLoader (!141)
- extracted an interface `ModuleProvider` from `ModuleLoader` and adjusted `LatexBuddy` and cli.py accordingly (!141)
- removed `LatexBuddy` methods `change_file` and `clear_error_list` and replaced their occurrences with `init` (!141)
- reimplemented highlighting algorithm enabling markings for different problems to overlap (!150)
- updated the Docker image to add TeX Live (!157)
- new logo (!173)

1.5.4 Fixed

- regex usage in own_checkers (!110)
- inconsistent naming of some checkers in config, Problem API and classnames (!108)
- shortened slightly lengthy methods in config_loader.py (!140)
- fixed critical bug in the highlighting system with reimplementation (!150)
- fixed bug in output.py which would break the HTML document, if a problem description contained linebreaks (!155)
- fixed tool loader so the modules directory can be anywhere on the file system (!159)

1.6 0.3.0 - 15 Jun 2021

1.6.1 Added

- centralized file for LaTeXBuddy exceptions (!94)
- checker to warn about low resolution in figures (!101)
- checker to detect \ref instead of e.g. \cref (!99)
- language support in whitelist for spelling or grammar errors (!102)
- added option to manually add keys and word lists to the whitelist via command line (!106)
- added Docker file for Docker-based install (!103)

1.6.2 Changed

- moved module execution time measurements from individual modules to the main buddy instance (!93)
- improved logging for tool-methods find_executable and execute_no_errors (!94)
- adapted all modules using tool-methods find_executable and execute_no_errors to the new features (!94)
- changed module execution to utilize multiprocessing (!92)
- changed Problem attribute position to be optional (!96)
- renamed Problem attribute cid to p_type and made it optional (!102)
- whitelist file extension removed (!102)

- number of suggestions in a problem is now capped at 10 (!102)

1.6.3 Fixed

- minor issue in languagetool.py: module didn't stop execution after java-check failed in find_languagetool_command() (!94)
- import issue with proselint, because proselint.py shared the same name with the imported API (!95)
- usage of old `compare_...` functions (#45, !97)
- whitelist working again (!102)
- invalid default value of cli flag `format` resulting in LaTeXBuddy ignoring the config option for `format` (#56, !104)

1.7 0.2.0 - 08 Jun 2021

1.7.1 Added

- button, to add to whitelist (!87)
- configuration files (!30)
- abstraction around the checked file using `TexFile` class (!45, !46)
- tool loader (!47)
- ability to select modules to be run (!48)
- CI job for publishing the package to the Registry (!51)
- error highlighting inside HTML output (!52)
- legal and copyright notices (!54)
- Proselint module (!60)
- various on-house modules
  - unreferenced figures (!65)
  - SiUnitX (!66, !67)
  - empty section (!68)
  - use of `\url` (!69)
- better logging (!73)
  - clearer, colourful console output
  - file log containing more verbose information
- verification options for config entries (!76)
- `--version`/`-v` option to the CLI (!83)
- in-file preprocessor for .tex files (!84)

1.7.2 Changed

- **BREAKING CHANGE:** minimal Python version set to 3.6.8 (!81)
- modules now adhere to the Abstract Module API (!22, !46)
- errors renamed to problems and now use new API (!42, !46)
- all modules that use the config now validate the config entries (!76)
- removed test_module.py (!77)
- improved spacing and sizing of the logo (!82)
- modules now adhere to the Abstract Module API (!22, !46)
- errors renamed to problems and now use new API (!42, !46)
- removed test_module.py (!77)

1.7.3 Fixed

- Aspell positions of problems in source file (!53, !55)
- HTML output not working properly with new APIs (!56)
- ChkTeX working incorrectly with text containing `:` (!70)
- Minor inconsistency in typing for Problem constructor's parameters (!75)
- unwanted spaces around problem text in HTML output (!80)

1.8 0.1.0 - 18 May 2021

This is the first (pre-)release of LaTeXBuddy.

1.8.1 Added

- basic project files (!1)
- main module functionality (!3)
- results output
  - HTML (!2, !25, !29)
  - JSON (!3)
- basic interoperability with non-Python checkers (!5, !6, !10)
- modules
  - LanguageTool (!3)
  - ChkTeX (!4)
  - Aspell (!5, !7, !8)
- whitelist (!21)
- tools

- command-line interface (!17)
- various development improvements
  - CI jobs for linting (!18) and smoke tests (!33)
- draft of the abstract module API (!22)
- logo (!38)

CHAPTER

TWO

LICENCE

2.1 This documentation

2.2 Source code

2.3 Copyright notices

LaTeXBuddy uses third-party software; refer to the NOTICE file for the complete list.

CHAPTER

THREE

USER'S GUIDE

3.1 Install

LaTeXBuddy is a Python package and thus can be installed with `pip`. However, it is not published to PyPI, but rather to GitLab Package Registry. This is because we do not want to publish our unfinished software just yet. You can expect LaTeXBuddy to become available on PyPI with the release of v1.0.0 Alpha 1.

3.1.1 Install from GitLab Package Registry

To install the package, execute the following command:

```
pip install latexbuddy --index-url https://gitlab.com/api/v4/projects/28436730/packages/
↪pypi/simple
```

This will install the latest version of the package.

Other versions

Important: GitLab Package Registry is mutable, which means we can delete packages if we want to. For now, we are experimenting with publishing a lot, so we can't guarantee that a particular version will never get deleted from the registry, especially the pre-release versions.

To install other versions and to view the generic information about the package, you can navigate to the package registry page.

3.2 Build from source

3.2.1 With Docker

Caution: The following is an experimental Docker build of LaTeXBuddy. It is not optimized and very unstable.

Prerequisites: Docker

The image is sadly not being distributed yet, so you have to build it yourself. It isn't complicated, but it takes around 7 minutes on a MacBook Pro and takes about 8 GB of extra space (the built container is around 1,15 GB). Once built, the image can be reused.

1. Build the image and tag it:

   ```
   docker build -t latexbuddy/latexbuddy .
   ```

2. To run the image once, run the following command:

   ```
   docker run --rm -v $(pwd):/latexbuddy latexbuddy/latexbuddy file_to_check.tex
   ```

   This will create a container, run the command on the file `file_to_check.tex` in your current directory. If you wish to set another directory as root, change `$(pwd)` to the desired path.

3. If you often check one file, you may want to create a container and run it without discarding it.

   1. First, create a container:

      ```
      docker create --name lb -v $(pwd):/latexbuddy latexbuddy/latexbuddy file_to_
      ↪check.tex
      ```

      The container will have the name `lb` — you are free to choose a different one.

   2. Every time you want to run checks, run:

      ```
      docker start -a lb
      ```

      The `-a` option redirects the output in your terminal, so you can see the output.

   3. After finishing, remove the container:

      ```
      docker rm lb
      ```

3.3 Developing a module

Having worked with LaTeXBuddy for some time, you may want to add a checking tool that is not part of the project yet. Fortunately, this is fairly easy thanks to LaTeXBuddy's focus on modularity.

3.3.1 Create a Python file for your module

Create a new .py file in `latexbuddy/modules/`. Within your file, add these import lines:

```
from typing import List

from latexbuddy.config_loader import ConfigLoader
from latexbuddy.texfile import TexFile
from latexbuddy.modules import Module
from latexbuddy.problem import Problem
```

You are now able to create a class inheriting from the abstract `Module` class which provides an API function for you to implement. Here is an *example* of how this could look like:

```
class MyNewModule(Module):

    def __init__(self):
        pass

    def run_checks(self, config: ConfigLoader, file: TexFile) -> List[Problem]:
        return []
```

Note: You are free to create as many other classes as needed, but remember that any class not inheriting from `Module` is ignored by LaTeXBuddy's module loader.

3.3.2 Working with `TexFile`

The `TexFile` class encapsulates all information about the LaTeX file that is supposed to be checked. It offers these attributes:

- `tex`: contains the contents of the `.tex` file as a String (`str`)
- `plain`: contains the contents of the deTeXed version (plain text) of the `.tex` file as a String (`str`)
- `tex_file`: contains the `.tex` file's path as a `pathlib.Path` object
- `plain_file`: contains the deTeXed version (plain text) of the `.tex` file as a `pathlib.Path` object
- `is_faulty`: contains a boolean that is `True`, if the `.tex` file is invalid or contains syntax errors and `False` otherwise

`TexFile` also offers two methods to convert positions in the deTeXed text to the corresponding positions in the original LaTeX code:

- `get_position_in_tex(char_pos: int) -> Optional[Tuple[int, int]]`: Takes in the absolute position of a character in the deTeXed text and returns the line and column of the same character in the original LaTeX code. If the specified position is invalid, None is returned.
- `get_position_in_tex_from_linecol(line: int, col: int) -> Optional[Tuple[int, int]]`: Takes in the line and column of a character in the deTeXed text and returns the line and column of the same character in the original LaTeX code. If the specified position is invalid, None is returned.

3.3.3 Working with `Problem`

The `Problem` class is a representation of a note/warning/error concerning a specific part of the text and is used as an interface between LaTeXBuddy and your module.
A `Problem` can be constructed with the following parameters:

position: Tuple[int, int] (optional)

A tuple specifying the problem's position in the checked `.tex` file and consists of two components: `(line_number, column_number)`. These numbers are referring to the position in the `.tex` file, NOT the deTeXed version. If no position is specified, the Problem is considered *general* and will appear in a different section than problems with a specific position.

Note: If you are checking the TeX version of the file and only have the absolute position of a problem, you can convert it using the first two return values of the *absolute_to_linecol* method in `latexbuddy.tools`.

Note: If you are checking the deTeXed version of the file, you need to convert the position of the problematic text in the deTeXed text into the position of the same text in the original LaTeX code using either the *get_position_in_tex* or the *get_position_in_tex_from_linecol* method provided by `TexFile`, depending on whether you are working with absolute positions or line, column tuples.

---

text: str (required)

A string containing the problematic part of the scanned text.

checker: Union[Type[NamedModule], NamedModule] (required)

A `Module` instance or the type of a checker inheriting from `Module` (this is used to ensure that module names stay consistent throughout LaTeXBuddy outputs).

file: pathlib.Path (required)

Attention: This is deprecated.

The path of the LaTeX file this problem refers to, given as a pathlib path.

p_type: Optional[str]

optional: A string containing an internal ID of the problem's category (e.g. 'double_whitespace' or 'missing_semicolon').

severity: ProblemSeverity = ProblemSeverity.WARNING

optional: an `Enum` specifying the level of severity for this problem. Valid values are:

- `NONE`: Problems are not being highlighted, but are still being output.
- `INFO`: Problems are highlighted with light blue color. These are suggestions; problems, that aren't criticizing the text. Example: suggestion to use "lots" instead of "a lot"
- `WARNING`: Problems are highlighted with orange color. These are warnings about problematic areas in documents. The files compile and work as expected, but some behavior may be unacceptable. Example: warning about using "$$" in LaTeX

- `ERROR`: Problems are highlighted with red color. These are errors, that prevent the documents to compile correctly. Example: not closed environment, or wrong LaTeX syntax

defaults to: `ProblemSeverity.WARNING`

category: Optional[str]

optional: a string containing the name of this problem's broader category, for example "grammar", "spelling" or "latex".

defaults to: `None`

description: Optional[str]

optional: a string containing a description of this problem or the reasoning behind it.

defaults to: `None`

context: Optional[Tuple[str, str]]

optional: the context of the problematic part of the text, given as a tuple containing the text before and after the problematic part. Although the size of the context is not restricted, it is recommended not to give considerably more context than the sentence that contains the problem.

defaults to: `None`

suggestions: List[str]

optional: suggestions to improve the problematic part of the text, given as a `List` of strings.

defaults to: `None`

key: Optional[str]

optional: a semi-unique string used to compare two problems (possibly from different checking tools). This is used primarily for whitelisting, so be as precise as needed, without being overly specific. It is recommended to start the key with the name of your new tool to ensure uniqueness among all checking tools.
If it's a pure **spelling tool** we recommend to put

```
key = "spelling" + "_" + errortext
```

as it allows for a more universal whitelist. If not you can also try to isolate the spelling errors and then set the key like above.

If not set you will **not** be able to whitelist your Problems!

defaults to: `None`

3.3.4 Further Information

For advanced information to improve the capabilities of your module and to make your life easier, feel free to read the manual on *Advanced API*. This page includes documentation for LaTeXBuddy's config and included utilities.

3.4 Using the API

New to LaTeXBuddy?

Please consider reading the *Beginners' Guide to Module development* first.

As you proceed developing your own module, you might want to simplify repeating processes and add some configuration options. Concerning that, LaTeXBuddy is offering its simple-to-use `ConfigLoader` and `tools` features.

3.4.1 Using the `ConfigLoader`

The `ConfigLoader` offers a simple way to configure LaTeXBuddy to your needs by providing support for a config file and integrating CLI flags.

Adding config options

LaTeXBuddy offers a default `config.py`, that can be tailored to your needs. To add your module and options to the `config.py`, follow these steps:

Add your module to `config.py`

To include your module into config, just add a new top-level entry into the `modules` dictionary consisting of your module class name as the `key` and an empty dictionary for the config options as the `value`.

Example:

```
main = {...}

modules = {
    "YourModuleClassName": {},
}
```

Add options for your module

As you want to add some config options for your module, that's the next step to complete. Just add your desired options to the empty dictionary created beforehand.

Example:

```
main = {...}

modules = {
    "YourModuleClassName": {
```

(continues on next page)

(continued from previous page)

```
        "sample_option": "sample_value",
        "meaning_of_life": 42,
    },
}
```

Note: As you may want to use LaTeXBuddy's enable/disable function, an `"enabled":True/False` entry needs to be added to your configuration.

Getting config options

Accessing configuration options generally requires two components: The first one is an instance or the type of a checker `Module`, or `None` for the configuration options of the main LaTeXBuddy instance. The second one is `key` which is essentially a string of your choosing that identifies a specific configuration option.

Config values can also be verified by providing a type, regex (for strings) or a list of possible values, which is handled via the parameters `verify_type`, `verify_regex` and `verify_choices`. If more than one verify parameter is specified, all specified requirements are checked. If a regex is provided, the `verify_type` parameter will always be set to `AnyStr` (even if another type was specified).

All configuration parameters are read from the config file that is specified in the Command Line call, but since CLI flags are translated to configuration options in `ConfigLoader` as well, they override any configuration option for the main LaTeXBuddy instance with the same key that might exist in the config file (e.g. "language", "output", "enable-modules-by-default" etc.).

`ConfigLoader` provides two functions for fetching configuration options:

`get_config_option(module, key, verify_type, verify_regex, verify_choices) -> Any`

This method fetches the value of the config entry with the specified key for the specified tool or raises a ConfigOptionNotFoundError, if such an entry doesn't exist.

Parameters:

- `module: Optional[Union[Type[NamedModule], NamedModule]]`: type or instance of the Module owning the config option
- `key: str`: key of the config option
- `verify_type: Type`: type that the config entry is required to be an instance of
- `verify_regex: Optional[str]`: regular expression that the config entry is required to match fully
- `verify_choices: Optional[Union[List[Any], Tuple[Any], Set[Any]]]`: a list/tuple/set of valid values in which the config entry is required to be contained

get_config_option(module, key, default_value, verify_type, verify_regex, verify_choices) -> Any

This method fetches the value of the config entry with the specified key for the specified tool or returns the specified default value, if such an entry doesn't exist.

Parameters:

- `module: Optional[Union[Type[NamedModule], NamedModule]]`: type or instance of the Module owning the config option
- `key: str`: key of the config option
- `default_value: Any`: default value in case the requested option doesn't exist
- `verify_type: Type`: type that the config entry is required to be an instance of
- `verify_regex: Optional[str]`: regular expression that the config entry is required to match fully
- `verify_choices: Optional[Union[List[Any], Tuple[Any], Set[Any]]]`: a list/tuple/set of valid values in which the config entry is required to be contained

3.4.2 Using the included utilities

LaTeXBuddy offers a variety of utility methods in `tools.py` which mainly include functions for finding and executing shell commands or python functions and converting character positions between absolute indexing and line, column tuples. The concrete functions are:

execute(*cmd: str, encoding: str) -> str

Executes a shell command via python's `subprocess` library and returns the combined contents of stdout and stderr as a string.

Parameters:

- `*cmd`: Tuple of strings representing the shell command and its flags and arguments
- *optional:* `encoding`: name of the encoding python uses to decode the contents in stdout and stderr

Example usage:

```
# execute command 'echo Hello World!' with tuple notation

execute("echo", "Hello", "World!")


# execute command 'echo Hello World!' with list notation
my_command = ["echo"]
my_command.append("Hello")
my_command.append("World!")

execute(*my_command)
```

`execute_background(*cmd: str) -> subprocess.Popen`

Executes a shell command in the background via python's `subprocess` library and returns a handle for the running process that can be used to terminate it with `kill_background_process`. Any output by the background process to stdout or stderr will be ignored.

Parameters:

- `*cmd`: Tuple of strings representing the shell command and its flags and arguments

`kill_background_process(process: subprocess.Popen) -> None`

Kills a previously started background process by sending a `SIGTERM` signal.

Parameters:

- `process`: Popen object representing a running process. Accepts return values of `execute_background`.

`execute_no_errors(*cmd: str, encoding: str = "ISO8859-1") -> str`

Executes a shell command via python's `subprocess` library and returns the contents of stdout as a string. Any output to stderr is ignored.

Parameters:

- `*cmd`: Tuple of strings representing the shell command and its flags and arguments
- *optional:* `encoding`: string name of the encoding python uses to decode the contents in stdout

`find_executable(name: str, to_install: Optional[str] = None, logger: Optional[Logger] = None, log_errors: bool = True) -> str`

Finds the path to a given executable with a call to `which`. Consequently, any executable that should be found must at least be in the user's `$PATH`. Raises a `FileNotFoundError`, if the executable could not be located.

Parameters:

- `name`: name of the executable to be found
- *optional:* `to_install`: correct name of the program or project which the requested executable belongs to (used in log messages, defaults to the value of `name`, if unspecified)
- *optional:* `logger`: logger instance of the calling module, defaults to the standard logger for tools.py
- *optional:* `log_errors`: specifies whether error messages should be logges as error (`True`) or debug (`False`) messages

`absolute_to_linecol(text: str, position: int) -> Tuple[int, int, List[int]]`

Calculates the line and column of a given character from the absolute position of that character in a specific text.

Parameters:

- `text`: text containing the character
- `position`: absolute position of the character (0-based)

`get_line_offsets(text: str) -> List[int]`

Calculates absolute character offsets for each line in the specified text and returns them as a list.

Indices correspond to the line numbers, but are 0-based. For example, if the first 4 lines contain 100 characters (including line breaks), `result[4]` will be 100. `result[0]` is always 0.

Parameters:

- `text`: the text to be processed

`is_binary(file_bytes: bytes) -> bool`

Detects whether the bytes of a file contain binary code or not. For correct detection, it is recommended, that at least 1024 bytes were read.

Parameters:

- `bytes`: bytes of a file

`execute_no_exceptions(function_call: Callable[[], None], error_message: str, traceback_log_level: Optional[str] = None) -> None`

Calls a function and catches any Exception that is raised during this. If an Exception is caught, the function is aborted and the error is logged, but as the Exception is caught, the program won't crash.

Parameters:

- `function_call`: python function to be executed
- *optional:* `error_message`: custom error message passed to the logger, defaults to `"An error occurred while executing lambda function"`
- *optional:* `traceback_log_level`: sets the log_level that is used to log the error traceback. If it is None, no traceback will be logged. Valid values are: "DEBUG", "INFO", "WARNING", "ERROR"

3.5 Command-line interface

3.5.1 Main executable

The one-stop-shop for LaTeX checking.

```
usage: latexbuddy [-h] [--version] [--verbose] [--config CONFIG]
                  [--output OUTPUT] [--language LANGUAGE]
                  [--whitelist WHITELIST]
                  [--format {HTML,html,JSON,json,HTML_FLASK,html_flask}]
                  [--enable-modules ENABLE_MODULES | --disable-modules DISABLE_MODULES]
                  file [file ...]
```

Positional Arguments

file	File(s) that will be processed.

Named Arguments

--version, -V	show program's version number and exit
--verbose, -v	Display debug output Default: 0
--config, -c	Location of the config file. Default: config.py
--output, -o	Directory, in which to put the output file.
--language, -l	Target language of the file.
--whitelist, -w	Location of the whitelist file.
--format, -f	Possible choices: HTML, html, JSON, json, HTML_FLASK, html_flask Format of the output file (either HTML or JSON).
--enable-modules	Comma-separated list of module names that should be executed. (Any other module will be implicitly disabled!)
--disable-modules	Comma-separated list of modules that should be disabled. (Every other module will be implicitly enabled!)

More documentation at <https://latexbuddy.readthedocs.io/>.

3.5.2 Whitelist operations

Perform whitelist operations.

```
usage: latexbuddy-whitelist [-h] [--whitelist WHITELIST]
                            {add,from-wordlist} ...
```

Named Arguments

--whitelist, -w	Location of the whitelist file. Default: ./whitelist

Sub-commands

add

Add keys to the whitelist.

```
latexbuddy-whitelist add [-h] KEY [KEY ...]
```

Positional Arguments

KEY	Keys that should be added to whitelist. Ideally, they should have been copied from the HTML output

from-wordlist

Read allowed words and add them to whitelist.

```
latexbuddy-whitelist from-wordlist [-h] WORD_LIST LANGUAGE
```

Positional Arguments

WORD_LIST	Path to the wordlist file. It should contain one word per line.
LANGUAGE	Language of the words in the wordlist

3.6 API Reference

3.6.1 Main instance

Contains the main LaTeXBuddy instance class.

class latexbuddy.buddy.**LatexBuddy**

The main instance of the applications that controls all the internal tools.

This is a singleton class with only one instance and exclusively static methods.

static add_error(*problem*)

Adds the error to the errors dictionary.

UID is used as key, the error object is used as value.

Parameters
problem (Problem) – problem to add to the dictionary

Return type
None

static add_to_whitelist(*uid*)

Adds the error identified by the given UID to the whitelist.

Afterwards this method deletes all other errors that are the same as the one just whitelisted.

Parameters
uid (*str*) – the UID of the error to be deleted

Return type
None

static check_whitelist()

Removes errors that are whitelisted.

Return type
None

static execute_module(*module*)

Executes checks for provided module and returns its Problems. This method is used to parallelize the module execution.

Parameters
module (*Module*) – module to execute

Returns
list of resulting problems

Return type
list[*latexbuddy.problem.Problem*]

static init(*config_loader*, *module_provider*, *file_to_check*, *path_list*, ***, *compile_tex*)

Initializes the LaTeXBuddy instance.

Parameters

- **config_loader** (ConfigLoader) – ConfigLoader object to manage config options
- **module_provider** (ModuleProvider) – ModuleProvider instance as a source of Module instances for running checks on the specified file
- **file_to_check** (*Path*) – file that will be checked
- **path_list** (*list[pathlib.Path]*) – a list of the paths for the html output
- **compile_tex** (*bool*) – boolean if the tex file should be compiled

Return type
None

static output_file()

Writes all current problems to the specified output file.

Return type
None

static output_html()

Renders all current problem objects as HTML and writes the file.

Return type
None

static output_json()

Writes all the current problem objects to the output file.

Return type
None

static run_tools()

Runs all modules in the LaTeXBuddy toolchain in parallel.

Return type
None

3.6.2 Configuration

This module describes the LaTeXBuddy config loader and its properties.

class latexbuddy.config_loader.**ConfigLoader**(*cli_arguments=None*)

Describes a ConfigLoader object.

The ConfigLoader processes LaTeXBuddy's cli arguments and loads the specified config file or the default config file, if none is specified. ConfigLoader also offers methods for accessing config entries with the option to specify a default value on Failure.

Parameters
cli_arguments (*Namespace | None*) –

get_config_option(*module*, *key*, *verify_type=typing.Any*, *verify_regex=None*, *verify_choices=None*)

This method fetches the value of the config entry with the specified key for the specified tool or raises a ConfigOptionNotFoundError, if such an entry doesn't exist or the retrieved entry does not match a specified verification criterion.

Parameters

- **module** (*Optional[Union[Type[NamedModule], NamedModule]]*) – type or an instance of the module owning the config option; if unspecified, this method will look for a configuration option in the main instance's dictionary
- **key** (*str*) – key of the config option
- **verify_type** (*Type*) – typing type that the config entry is required to be an instance of (otherwise ConfigOptionVerificationError is raised)
- **verify_regex** (*str | None*) – regular expression that the config entry is required to match fully (otherwise ConfigOptionVerificationError is raised)

  Note: this overrides verify_type with 'AnyStr'
- **verify_choices** (*List[Any] | Tuple[Any] | Set[Any] | None*) – a list/tuple/set of valid values in which the config entry needs to be contained in order to be valid

Returns
the value of the requested config option, if it exists

Raises
ConfigOptionNotFoundError, if the requested config option doesn't exist

Raises
ConfigOptionVerificationError, if the requested config option does not meet the specified criteria

Return type
Any

`get_config_option_or_default`(*module*, *key*, *default_value*, *verify_type=typing.Any*, *verify_regex=None*, *verify_choices=None*)

This method fetches the value of the config entry with the specified key for the specified tool or returns the specified default value, if such an entry doesn't exist or the retrieved entry does not match a specified verification criterion.

Parameters

- **module** (*Optional[Union[Type[NamedModule], NamedModule]]*) – type or an instance of the module owning the config option; if unspecified, this method will look for a configuration option in the main instance's dictionary
- **key** (*str*) – key of the config option
- **default_value** (*Any*) – default value in case the requested option doesn't exist
- **verify_type** (*Type*) – typing type that the config entry is required to be an instance of (otherwise `ConfigOptionVerificationError` is raised)
- **verify_regex** (*str | None*) – regular expression that the config entry is required to match fully (otherwise `ConfigOptionVerificationError` is raised).

  Note: this overrides `verify_type` with `typing.AnyStr`
- **verify_choices** (*List[Any] | Tuple[Any] | Set[Any] | None*) – a list/tuple/set of valid values in which the config entry needs to be contained in order to be valid

Returns
the value of the requested config option or `default_value`, if the config option doesn't exist

Return type
Any

`load_configurations`(*config_file_path*)

This helper-function loads the contents of a specified config.

.py- file.

Parameters
config_file_path (*Path*) – config file to be loaded (.py)

Returns
None

Return type
None

3.6.3 TeX file

This module defines new TexFile class used to abstract files LaTeXBuddy is working with.

`class latexbuddy.texfile.`**TexFile**(*file*, ***, *compile_tex*)

A simple TeX file.

This class reads the file, detects its encoding and saves it as text for future editing.

Parameters

- **file** (*Path*) –
- **compile_tex** (*bool*) –

get_position_in_tex(*char_pos*)

Gets position of a character in the original file.

Parameters
char_pos (*int*) – absolute char position

Returns
line and column number of the respective char in the tex file

Return type
tuple[int, int] | None

3.6.4 Modules

class latexbuddy.module_loader.**ModuleLoader**(*directory*)

This class encapsulates all features necessary to load LaTeXBuddy modules from a specified directory.

Parameters
directory (*Path*) –

find_py_files()

This method finds all .py files within the ModuleLoader's directory or any subdirectories and returns a list of their paths.

Returns
a list of all Python files in the ModuleLoader's directory (or subfolders)

Return type
list[pathlib.Path]

import_py_files()

This method loads a python module from the specified file path for a list of file paths.

Returns
a list of python modules ready to be used

Return type
list[module]

load_modules()

This method loads every module that is found in the ModuleLoader's directory.

Returns
a list of instances of classes implementing the Module API

Return type
list[latexbuddy.modules.Module]

load_selected_modules(*cfg*)

This method loads every module that is found in the ModuleLoader's directory and only returns instances of modules that are enabled in the specified configuration context.

Parameters
cfg (ConfigLoader) – ConfigLoader instance containing config options for enabled/disabled tools

Returns
a list of instances of classes implementing the Module API which have been enabled in the specified configuration context

Return type
list[latexbuddy.modules.Module]

class latexbuddy.module_loader.**ModuleProvider**

This interface class defines all methods necessary to provide a list of instances of modules that implement the Module API, which is required in order for the instances to be executed by the main LatexBuddy instance.

abstract load_selected_modules(*cfg*)

This method loads every module that is found in the ModuleLoader's directory and only returns instances of modules that are enabled in the specified configuration context.

Parameters
cfg (ConfigLoader) – ConfigLoader instance containing config options for enabled/disabled tools

Returns
a list of instances of classes implementing the Module API which have been enabled in the specified configuration context

Return type
list[latexbuddy.modules.Module]

3.6.5 Problems

This module describes the LaTeXBuddy Problem class and its properties.

Problems are found by *Checkers*. *Checkers* are free to implement their own *Problem* types, however, LaTeXBuddy will most surely not display extra metadata.

class latexbuddy.problem.**Problem**(*position*, *text*, *checker*, *file*, *severity=ProblemSeverity.WARNING*, *p_type=None*, *length=None*, *category=None*, *description=None*, *context=None*, *suggestions=None*, *key=None*)

Describes a Problem object.

A Problem object contains information about a problem detected by a checker. For example, it can be wrong LaTeX code or a misspelled word.

Parameters

- **position** (*tuple[int, int] | None*) –
- **text** (*str*) –
- **checker** (*type[NamedModule] | NamedModule*) –
- **file** (*Path*) –
- **severity** (ProblemSeverity) –
- **p_type** (*str | None*) –
- **length** (*int | None*) –
- **category** (*str | None*) –
- **description** (*str | None*) –
- **context** (*tuple[str, str] | None*) –

- **suggestions** (*list[str] | None*) –
- **key** (*str | None*) –

better_eq(*key*)

equal method based on the key/CompareID.

Parameters
key (*str*) –

Return type
bool

class latexbuddy.problem.**ProblemJSONEncoder**(**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, sort_keys=False, indent=None, separators=None, default=None*)

Provides JSON serializability for class Problem.

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

Parameters
obj (*Any*) –

Return type
dict[str, *Any*]

class latexbuddy.problem.**ProblemSeverity**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Defines possible problem severity grades.

Problem severity is usually preset by the checkers themselves. However, a user should be able to redefine the severity of a specific problem, using either `category`, `key`, or `p_type`.

- `"none"` problems are not being highlighted, but are still being output.
- `"info"` problems are highlighted with light blue colour. These are suggestions; problems, that aren't criticising the text. Example: suggestion to use "lots" instead of "a lot"
- `"warning"` problems are highlighted with orange colour. These are warnings about problematic areas in documents. The files compile and work as expected, but some behaviour may be unacceptable. Example: warning about using "$$" in LaTeX
- `"error"` problems are highlighted with red colour. These are errors, that prevent the documents to compile correctly. Example: not closed environment, or plain wrong LaTeX syntax

latexbuddy.problem.**set_language**(*lang*)

Sets the static variable language used for key generation.

Parameters
lang (*str | None*) – global language that the modules currently work with

Return type
None

3.6.6 Output

class latexbuddy.output.**Interval**(*problems*, *start=None*, *end=None*)

This class describes an interval with problems.

An interval is a section of text, defined byt its start and end positions, that contains *Problem* objects.

Parameters

- **problems** (Problem | *list[*Problem*]*) – list of problems on the interval
- **start** (*int | None*) – start symbol of the interval
- **end** (*int | None*) – end symbol of the interval

intersects(*other*)

Determines whether or not the other interval intersects with 'self'.

Parameters
other (Interval) – other interval to consider

Return type
bool

perform_intersection(*other*)

Performs an intersection of two intervals and returns a list of new non-intersecting intervals to replace the two specified intervals 'self' and 'other', if the intervals actually intersect. Should the intervals not intersect, None is returned, indicating that there is no need to replace the two intervals. The intervals in the returned list are sorted by their start index in ascending order.

Parameters
other (Interval) – other interval to intersect with 'self'

Return type
list[*latexbuddy.output.Interval*] | None

latexbuddy.output.**add_basic_problem_intervals**(*line_intervals*, *problems*, *tex_lines*)

Filters out problems without a position attribute or with length zero and inserts the remaining ones into the line_intervals list.

Parameters

- **line_intervals** (*list[list[*latexbuddy.output.Interval*]]*) – List of lists of Intervals for any given line
- **problems** (*list[*latexbuddy.problem.Problem*]*) – list of problems to be inserted as Intervals
- **tex_lines** (*list[str]*) – contents of the .tex-file

Return type
None

latexbuddy.output.**create_empty_line_interval_list**(*lines*)

Creates and returns a list of (empty) lists of Intervals. The outer list will contain exactly len(tex_lines) + 1 empty lists.

Parameters
lines (*list[str]*) – individual lines of a .tex-file as a list of strings

Returns
a list of empty lists that meet the specified dimensions

Return type
list[list[*latexbuddy.output.Interval*]]

latexbuddy.output.**generate_wrapper_html_tags**(*interval*)

Generates and returns a pair of HTML tags to wrap the text in the specified interval.

Parameters
interval (Interval) – interval, specifying the position and metadata of the tags

Returns
a tuple of two strings, containing an opening and a closing tag for the specified interval object

Return type
tuple[str, str]

latexbuddy.output.**highlight**(*tex*, *problems*)

Highlights the TeX code using the problems' data.

Parameters

- **tex** (*str*) – TeX source
- **problems** (*list[latexbuddy.problem.Problem]*) – list of problems

Returns
HTML string with highlighted errors, ready to be put inside <pre>

Return type
str

latexbuddy.output.**mark_intervals_in_tex**(*lines*, *line_intervals*)

Adds HTML marker-tags for every interval in multiple lines of TeX code.

For every line in lines, and for every interval in line_intervals for the respective line, this method wraps it with tags and returns the resulting line. This method also escapes all HTML control characters included in tex_line.

It basically calls *mark_intervals_in_tex_line()*, but the lines are modified in-place.

Parameters

- **lines** (*list[str]*) – lines from the TeX file
- **line_intervals** (*list[list[latexbuddy.output.Interval]]*) – list of non-intersecting intervals to be highlighted for every line

Return type
None

latexbuddy.output.**mark_intervals_in_tex_line**(*line*, *intervals*)

Adds HTML marker-tags for every interval in a line of TeX code.

For every interval in `intervals`, this method wraps it with tags and returns the resulting line. This method also escapes all HTML control characters included in `tex_line`.

Parameters

- **line** (*str*) – line from the TeX file
- **intervals** (*list[latexbuddy.output.Interval]*) – list of non-intersecting intervals to be highlighted in the line

Returns
resulting line as a string, containing tags

Return type
str

latexbuddy.output.**problem_key**(*problem*)

Returns a number for each problem to be able to sort them.

This puts YaLaFi's problems on top, followed by errors without location.

Parameters
problem (Problem) – problem object

Returns
error's "rating" for sorting

Return type
int

latexbuddy.output.**render_general_html**(*template*, *file_name*, *file_text*, *problems*, *path_list*, *pdf_path*)

Renders an HTML page based on file contents and discovered problems.

Parameters

- **template** (*Template*) – HTML template to use for generation
- **file_name** (*str*) – file name
- **file_text** (*str*) – contents of the file
- **problems** (*dict[str, latexbuddy.problem.Problem]*) – dictionary of errors returned from latexbuddy
- **pdf_path** (*str*) – path of pdf file
- **path_list** (*Path*) – a list, containing all file paths to the checked files

Returns
generated HTML

Return type
str

latexbuddy.output.**resolve_interval_intersections**(*intervals*)

Finds any intersecting intervals and replaces them with non- intersecting intervals that may contain more than one problem.

Parameters
intervals (*list[latexbuddy.output.Interval]*) – list of intervals in one line to be checked for intersections

Return type
None

3.6.7 Preprocessing

class latexbuddy.preprocessor.**LineProblemFilter**(*start_line*, *end_line=None*)

ProblemFilter implementation that only considers a problem's line position.

Parameters

- **start_line** (*int*) –
- **end_line** (*int | None*) –

custom_match(*problem*)

Matches a given Problem object based on custom parameters of the subclass implementation.

Parameters
problem (Problem) – Problem object to be examined

Returns
True, if the problem matches all custom requirements, False otherwise

Return type
bool

custom_parameters_equal(*other*)

Determines, if two custom ProblemFilter objects are equal.

Two objects of type *ProblemFilter* are considered equal as long as they are:

- of the same type
- equal in terms of their custom parameters

> **Caution:** This method does not check the equality of *start_line* ` and end_line!

Parameters
other (ProblemFilter) – second custom ProblemFilter to be compared with the current one

Returns
True, if the other custom ProblemFilter is equal to the current one, False otherwise

Return type
bool

class latexbuddy.preprocessor.**ModuleProblemFilter**(*module_name*, *start_line*, *end_line=None*)

ProblemFilter implementation that filters problems, if they have been created by a specified LaTeXBuddy module.

Parameters

- **module_name** (*str*) –
- **start_line** (*int*) –
- **end_line** (*int | None*) –

custom_match(*problem*)

Matches a given Problem object based on custom parameters of the subclass implementation.

Parameters
problem (Problem) – Problem object to be examined

Returns
True, if the problem matches all custom requirements, False otherwise

Return type
bool

custom_parameters_equal(*other*)

Determines, if two custom ProblemFilter objects are equal.

Two objects of type *ProblemFilter* are considered equal as long as they are:

- of the same type
- equal in terms of their custom parameters

> **Caution:** This method does not check the equality of *start_line* ` and end_line!

Parameters
other (ProblemFilter) – second custom ProblemFilter to be compared with the current one

Returns
True, if the other custom ProblemFilter is equal to the current one, False otherwise

Return type
bool

class latexbuddy.preprocessor.**Preprocessor**

This class represents the LaTeXBuddy in-file preprocessor.

the Preprocessor is capable of parsing buddy commands disguised as LaTeX comments from a TexFile object using regexes and is able to filter any given Problem or list of Problems accordingly.

apply_preprocessor_filter(*problems*)

Applies all parsed ProblemFilters and returns all non- matching Problems.

Parameters
problems (*list*[latexbuddy.problem.Problem]) – list of Problems to filter

Returns
filtered list of Problems

Return type
list[*latexbuddy.problem.Problem*]

matches_preprocessor_filter(*problem*)

Checks, if the provided Problem matches any filter.

Parameters
problem (Problem) – Problem to check

Returns
false if matching; true otherwise

Return type
bool

`regex_parse_preprocessor_comments`(*file*)

Parses preprocessor statements in a TeX file.

This method takes a *TexFile* object and parses all preprocessor statements contained in it. This results in a set of of *ProblemFilter* objects, which are then added to this instance's list of filters and later applied to the problems.

Parameters
file (TexFile) – TeX file object containing the LaTeX source code to be parsed

Return type
None

`class latexbuddy.preprocessor.ProblemFilter`(*start_line*, *end_line=None*)

Describes the base class for any problem filter.

`ProblemFilter` provides functionality to define a start and end line and to match a *Problem* based on its line position.

`ProblemFilter` objects can be made "open-ended" by omitting the `end_line` parameter. This results in a filter matching any problem located at or below the `start_line`. Open-ended filters can later be closed by supplying the `end_line` via the *end()* method.

For more diverse filters, ProblemFilter provides the following abstract methods which must be implemented by all subclasses: *custom_match()* and *custom_parameters_equal()*.

Parameters

- **start_line** (*int*) – beginning of the filter's area
- **end_line** (*int | None*) – end of the filter's area (open-ended, if omitted)

`abstract custom_match`(*problem*)

Matches a given Problem object based on custom parameters of the subclass implementation.

Parameters
problem (Problem) – Problem object to be examined

Returns
`True`, if the problem matches all custom requirements, `False` otherwise

Return type
bool

`abstract custom_parameters_equal`(*other*)

Determines, if two custom `ProblemFilter` objects are equal.

Two objects of type *ProblemFilter* are considered equal as long as they are:

- of the same type
- equal in terms of their custom parameters

> **Caution:** This method does not check the equality of *start_line* ` and `end_line`!

Parameters
other (ProblemFilter) – second custom ProblemFilter to be compared with the current one

Returns
True, if the other custom ProblemFilter is equal to the current one, False otherwise

Return type
bool

end(*end_line*)

Sets the end line of ProblemFilter, if not already done.

Parameters
end_line (*int*) – line number of the filter's end

Returns
True if end_line was set before; False otherwise

Return type
bool

match(*problem*)

Matches custom filter's requirements against a problem.

This method determines, whether a given problem is located within the filter's line boundaries and matches all custom requirements that the subclass implementation imposes.

Parameters
problem (Problem) – Problem object to examine

Returns
True, if the problem is located in the area covered by the ProblemFilter and matches all custom requirements, False otherwise

Return type
bool

class latexbuddy.preprocessor.**SeverityProblemFilter**(*severity*, *start_line*, *end_line=None*)

ProblemFilter implementation that filters problems, if they have been created with a specified ProblemSeverity.

Parameters

- **severity** (ProblemSeverity) –
- **start_line** (*int*) –
- **end_line** (*int | None*) –

custom_match(*problem*)

Matches a given Problem object based on custom parameters of the subclass implementation.

Parameters
problem (Problem) – Problem object to be examined

Returns
True, if the problem matches all custom requirements, False otherwise

Return type
bool

`custom_parameters_equal`(*other*)

Determines, if two custom `ProblemFilter` objects are equal.

Two objects of type *ProblemFilter* are considered equal as long as they are:

- of the same type
- equal in terms of their custom parameters

> **Caution:** This method does not check the equality of *start_line* ` and `end_line`!

Parameters
other (ProblemFilter) – second custom ProblemFilter to be compared with the current one

Returns
`True`, if the other custom ProblemFilter is equal to the current one, `False` otherwise

Return type
bool

`class latexbuddy.preprocessor.`**`WhitelistKeyProblemFilter`**(*wl_key*, *start_line*, *end_line=None*)

This filter excludes problems, if they have been created with a specified whitelist key.

Parameters

- **wl_key** (*str*) – whitelist key of a problem
- **start_line** (*int*) – beginning of the filter's area
- **end_line** (*int | None*) – end of the filter's area

`custom_match`(*problem*)

Matches a given Problem object based on custom parameters of the subclass implementation.

Parameters
problem (Problem) – Problem object to be examined

Returns
`True`, if the problem matches all custom requirements, `False` otherwise

Return type
bool

`custom_parameters_equal`(*other*)

Determines, if two custom `ProblemFilter` objects are equal.

Two objects of type *ProblemFilter* are considered equal as long as they are:

- of the same type
- equal in terms of their custom parameters

> **Caution:** This method does not check the equality of *start_line* ` and `end_line`!

Parameters
other (ProblemFilter) – second custom ProblemFilter to be compared with the current one

Returns
`True`, if the other custom ProblemFilter is equal to the current one, `False` otherwise

Return type
bool

3.6.8 Whitelist

`latexbuddy.whitelist.`**add_to_whitelist**(*whitelist*, *keys*)

Adds a list of keys to the whitelist.

Keys should be valid keys, ideally copied from LaTeXBuddy HTML output.

Parameters

- **whitelist** (*Path*) – path to the whitelist file
- **keys** (*list[str]*) – list of keys

Return type
None

`latexbuddy.whitelist.`**fill_whitelist_from_wordlist**(*whitelist*, *wordlist*, *language*)

Adds keys to the whitelist based on a list of words.

Words in the wordlist should all be from the same language. Each line should be a single word.

Parameters

- **whitelist** (*Path*) – path to the whitelist file
- **wordlist** (*Path*) – path to the wordlist file
- **language** (*str*) – language of the words in the wordlist

Return type
None

3.6.9 Utilities

Exceptions

This module defines standard exceptions that are to be raised when certain application-specific errors occur.

exception `latexbuddy.exceptions.`**ConfigOptionError**

Base Exception for errors related to loading configurations.

exception `latexbuddy.exceptions.`**ConfigOptionNotFoundError**

Describes a ConfigOptionNotFoundError.

This error is raised when a requested config entry doesn't exist.

exception `latexbuddy.exceptions.`**ConfigOptionVerificationError**

Describes a ConfigOptionVerificationError.

This error is raised when a requested config entry does not meet the specified criteria.

exception `latexbuddy.exceptions.`**ExecutableNotFoundError**

This error is raised when LaTeXBuddy can not locate a third-party executable dependency on the system it is running on.

exception latexbuddy.exceptions.**LanguageNotSupportedError**

This error is raised when LaTeXBuddy or a submodule does not support the configured language.

Messages

This module defines standard messages that are to be printed to the command line as well as builders for those.

Tools

This module contains various utility tools.

latexbuddy.tools.**absolute_to_linecol**(*text*, *position*)

Calculates line and column number for an absolute character position.

Parameters

- **text** (*str*) – text of file to find line:col position for
- **position** (*int*) – absolute 0-based character position

Returns
line number, column number, line offsets

Return type
tuple[int, int, list[int]]

class latexbuddy.tools.**classproperty**(*fget=None*, *fset=None*, *fdel=None*, *doc=None*)

Provides a way to implement a python property with class-level accessibility.

latexbuddy.tools.**execute**(**cmd*, *encoding='ISO8859-1'*)

Executes a terminal command with subprocess.

See usage example in latexbuddy.aspell.

Parameters

- **cmd** (*str*) – command name and arguments
- **encoding** (*str*) – output encoding

Returns
command output

Return type
str

latexbuddy.tools.**execute_background**(**cmd*)

Executes a terminal command in background.

Parameters
cmd (*str*) – command name and arguments

Returns
subprocess instance of the executed command

Return type
Popen[bytes]

latexbuddy.tools.**execute_no_errors**(**cmd*, *encoding='ISO8859-1'*)

Executes a terminal command while suppressing errors.

Parameters

- **cmd** (*str*) – command name and arguments
- **encoding** (*str*) – output encoding

Returns
command output

Return type
str

latexbuddy.tools.**execute_no_exceptions**(*function_call*, *error_message='An error occurred while executing lambda function'*, *traceback_log_level=None*)

Calls a function and catches any Exception that is raised during this.

If an Exception is caught, the function is aborted and the error is logged, but as the Exception is caught, the program won't crash.

Parameters

- **function_call** (*Callable[[], None]*) – function to be executed
- **error_message** (*str*) – custom error message displayed in the console
- **traceback_log_level** (*str | None*) – sets the log_level that is used to log the error traceback. If it is None, no traceback will be logged. Valid values are: "DEBUG", "INFO", "WARNING", "ERROR"

Return type
None

latexbuddy.tools.**find_executable**(*name*, *to_install=None*, *err_logger=<Logger latexbuddy.tools (DEBUG)>*, ***, *log_errors=True*)

Finds path to an executable.

If the executable can not be located, an error message is logged to the specified logger, otherwise the executable's path is logged as a debug message.

This uses 'which', i.e. the executable should at least be in user's $PATH

Parameters

- **name** (*str*) – executable name
- **to_install** (*str | None*) – correct name of the program or project which the requested executable belongs to (used in log messages)
- **err_logger** (*Logger*) – custom logger to be used for logging debug/error messages
- **log_errors** (*bool*) – specifies whether or not this method should log an error message, if the executable can not be located; if this is False, a debug message will be logged instead

Returns
path to the executable

Raises
FileNotFoundError – if the executable couldn't be found

Return type
str

latexbuddy.tools.**get_all_paths_in_document**(*file_path*)

Checks files that are included in a file.

If the file includes more files, these files will also be checked.

:param file_path:a string, containing file path :return: the files to check

Parameters
file_path (*Path*) –

Return type
list[pathlib.Path]

latexbuddy.tools.**get_command_string**(*cmd*)

Constructs a command string from a tuple of arguments.

Parameters
cmd (*tuple[str, ...]*) – tuple of command line arguments

Returns
the command string

Return type
str

latexbuddy.tools.**get_line_offsets**(*text*)

Calculates character offsets for each line in the file.

Indices correspond to the line numbers, but are 0-based. For example, if first 4 lines contain 100 characters (including line breaks), *result[4]* will be 100. *result[0]* = 0.

This is a port of YaLaFi's get_line_starts() function.

Parameters
text (*str*) – contents of file to find offsets for

Returns
list of line offsets with indices representing 0-based line numbers

Return type
list[int]

latexbuddy.tools.**is_binary**(*file_bytes*)

Detects whether the bytes of a file contain binary code or not.

For correct detection, it is recommended, that file_bytes is at least 1024 bytes long.

Sources:

- https://stackoverflow.com/a/7392391/4735420
- https://github.com/file/file/blob/f2a6e7cb7d/src/encoding.c#L151-L228

Parameters
file_bytes (*bytes*) – bytes of a file

Returns
True, if the file is binary, False otherwise

Return type
bool

latexbuddy.tools.**kill_background_process**(*process*)

Kills previously opened background process.

For example, it can accept the return value of *execute_background()* as argument.

Parameters
process (*Popen*) – subprocess instance of the process

Return type
None

latexbuddy.tools.**match_lines**(*lines*, *unchecked_files*, *checked_files*)

Matches the lines with the given regexes.

Parameters

- **lines** (*list[str]*) – the lines
- **unchecked_files** (*list[pathlib.Path]*) – the unchecked_files
- **checked_files** (*list[pathlib.Path]*) – the checked_files

Returns
the unchecked_files

Return type
list[pathlib.Path]

3.7 Built-in modules

3.7.1 aspell

This module defines the connection between LaTeXBuddy and GNU Aspell.

class latexbuddy.modules.aspell.**Aspell**

static find_languages()

Returns all languages supported by the current aspell installation. Omits specific language variations like 'en- variant_0'.

Returns
list of supported languages in str format

Return type
list[str]

format_errors(*out*, *line_number*, *file*)

Parses Aspell errors and returns list of Problems.

Parameters

- **line_number** (*int*) – the line_number for the location
- **out** (*list[str]*) – line-split output of the aspell command
- **file** (*TexFile*) – the file path

Return type
list[*latexbuddy.problem.Problem*]

run_checks(*config*, *file*)

Runs the Aspell checks on a file and returns the results as a list.

Requires Aspell to be set up.

Parameters

- **config** (ConfigLoader) – the configuration options of the calling LaTeXBuddy instance
- **file** (TexFile) – LaTeX file to be checked (with built-in detex option)

Return type
list[*latexbuddy.problem.Problem*]

3.7.2 BibTeX

class latexbuddy.modules.bib_checkers.**BibtexDuplicates**

run_checks(*config*, *file*)

Runs the checks and returns a list of discovered problems.

Parameters

- **config** (ConfigLoader) – the configuration options of the calling LaTeXBuddy instance
- **file** (TexFile) – LaTeX file to be checked (with built-in detex option)

Return type
list[*latexbuddy.problem.Problem*]

class latexbuddy.modules.bib_checkers.**NewerPublications**

run_checks(*config*, *file*)

Runs the checks and returns a list of discovered problems.

Parameters

- **config** (ConfigLoader) – the configuration options of the calling LaTeXBuddy instance
- **file** (TexFile) – LaTeX file to be checked (with built-in detex option)

Return type
list[*latexbuddy.problem.Problem*]

latexbuddy.modules.bib_checkers.**get_bibfile**(*file*)

Checks the given file text for a ibliography{} command and returns the full path of the input BibTeX file. For now only works with a single BibTeX file.

Parameters
file (TexFile) – TexFile object of the LaTeX file to check

Returns

Return type
Path | None

latexbuddy.modules.bib_checkers.**parse_bibfile**(*bibfile*)

Parses the given BibTeX file to extract the publications.

Parameters
bibfile (*Path*) – Path object of the BibTeX file to be parsed

Returns
the title, year, and BibTeX Id of each publication as 3-Tuple

Return type
list[tuple[str, str, str]]

3.7.3 ChkTeX

This module defines the connection between LaTeXBuddy and ChkTeX.

ChkTeX Documentation: https://www.nongnu.org/chktex/ChkTeX.pdf

3.7.4 Diction

class latexbuddy.modules.diction.**Diction**

format_errors(*out*, *original*, *file*)

Parses diction errors and returns list of Problems.

Parameters

- **original** (*list[str]*) – lines of file to check as list
- **out** (*list[str]*) – line split output of the diction command with empty lines removed
- **file** (TexFile) – the TexFile instance

Return type
list[*latexbuddy.problem.Problem*]

run_checks(*config*, *file*)

Runs the checks and returns a list of discovered problems.

Parameters

- **config** (ConfigLoader) – the configuration options of the calling LaTeXBuddy instance
- **file** (TexFile) – LaTeX file to be checked (with built-in detex option)

Return type
list[*latexbuddy.problem.Problem*]

3.7.5 LanguageTool

This module defines the connection between LaTeXBuddy and LanguageTool.

class latexbuddy.modules.languagetool.**LanguageTool**

Wraps the LanguageTool API calls to check files.

check_tex(*file*)

Runs the LanguageTool checks on a file.

Parameters
file (TexFile) – the file to run checks on

Return type
list[*latexbuddy.problem.Problem*]

execute_commandline_request(*file*)

Execute the LanguageTool command line app to check the text.

Parameters
file (TexFile) – TexFile object representing the file to be checked

Returns
app's response

Return type
dict[str, *Any*]

find_disabled_rules(*config*)

Reads all disabled rules and categories from the specified configuration and saves the result in the instance.

Parameters
config (ConfigLoader) – configuration options to be read

Return type
None

find_languagetool_command()

Searches for the LanguageTool command line app.

This method also checks if Java is installed.

Return type
None

find_languagetool_command_prefix()

Finds the prefix of the shell command executing LanguageTool in the commandline.

Return type
list[str]

find_supported_languages()

Acquires a list of supported languages from the version of LanguageTool that is currently used.

Return type
list[str]

static format_errors(*raw_problems*, *file*)

Parses LanguageTool errors and converts them to LaTeXBuddy Error objects.

Parameters

- **raw_problems** (*dict[str, Any]*) – LanguageTool's error output
- **file** (TexFile) – TexFile object representing the file to be checked

Return type
list[*latexbuddy.problem.Problem*]

lt_languages_get_request(*server_url*)

Sends a GET request to the specified URL in order to retrieve a JSON formatted list of supported languages by the server.

If the response format is invalid, this method will most likely fail with an exception.

Parameters
server_url (*str*) –

Return type
list[str]

lt_post_request(*file*, *server_url*)

Send a POST request to the LanguageTool server to check the text.

Parameters

- **file** (TexFile) – TexFile object representing the file to be checked
- **server_url** (*str*) – URL of the LanguageTool server

Returns
server's response

Return type
dict[str, *Any*]

matches_language_regex(*language*)

Determines whether a given string is a language code by matching it against a regular expression.

Parameters
language (*str*) –

Return type
bool

static parse_error_replacements(*json_replacements*, *max_elements=5*)

Converts LanguageTool's replacements to LaTeXBuddy suggestions list.

Parameters

- **json_replacements** (*list[dict[str, Any]]*) – list of LT's replacements for a particular word
- **max_elements** (*int*) – max amount of proposed replacements for the given error

Returns
list of string values of said replacements

Return type
list[str]

run_checks(*config*, *file*)

Runs the LanguageTool checks on a file and returns the results as a list.

Requires LanguageTool (server) to be set up. Local or global servers can be used.

Parameters

- **config** (ConfigLoader) – the configuration options of the calling LaTeXBuddy instance
- **file** (TexFile) – LaTeX file to be checked (with built-in detex option)

Return type
list[*latexbuddy.problem.Problem*]

class latexbuddy.modules.languagetool.**LanguageToolLocalServer**

Defines an instance of a local LanguageTool deployment.

static **find_free_port**(*port=None*)

Tries to find a free port for the LanguageTool server.

Parameters
port (*int | None*) – port to check first

Returns
a free port that the LanguageTool server can listen at

Return type
int

get_server_run_command()

Searches for the LanguageTool server executable.

This method also checks if Java is installed.

Return type
None

static **is_port_in_use**(*port*)

Checks if a port is already in use.

Parameters
port (*int*) – port to check

Returns
True if port already in use, False otherwise

Return type
bool

start_local_server(*port=8081*)

Starts the LanguageTool server locally.

Parameters
port (*int*) – port for the server to listen at

Returns
the actual port of the server

Return type
int

stop_local_server()

Stops the local LanguageTool server process.

Return type
None

wait_till_server_up()

Waits for the LanguageTool server to start.

Raises
ConnectionError – if server didn't start

Return type
None

class latexbuddy.modules.languagetool.**Mode**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Describes the LanguageTool mode.

LanguageTool can be run as a command line program, a local server, or a remote server.

3.7.6 Log filter

class latexbuddy.modules.logfilter.**LogFilter**

A Filter for log files.

Using TexFilt: https://www.ctan.org/tex-archive/support/texfilt

format_problems(*raw_problems_path*, *file*)

Formats the output to a List of Problems.

Parameters

- **raw_problems_path** (*Path*) – Path to TexFilt output
- **file** (TexFile) – file to check

Returns
a list of problems

Return type
list[*latexbuddy.problem.Problem*]

run_checks(*config*, *file*)

Runs the Texfilt checks on a file and returns the results as a list.

Parameters

- **config** (ConfigLoader) – configurations of the LaTeXBuddy instance
- **file** (TexFile) – the file to run checks on

Returns
a list of problems

Return type
list[*latexbuddy.problem.Problem*]

3.7.7 Own checkers

3.7.8 Proselint

3.7.9 Yalafi

3.8 Server API Reference

class latexbuddy.flask_app.**FlaskConfigLoader**(*output_dir*, *language*, *module_selector_mode*, *module_selection*, *whitelist_id*)

Parameters

- **output_dir** (*Path*) –
- **language** (*str | None*) –
- **module_selector_mode** (*str | None*) –
- **module_selection** (*str | None*) –
- **whitelist_id** (*str | None*) –

CHAPTER

FOUR

DEVELOPER'S GUIDE

4.1 Environment setup

4.1.1 OS

You can code on any system you want! However, since the project is in the first place targeted at the user of Unix-like systems, we highly recommend you install on. Here are some user-friendly distributions for you to try out:

- Ubuntu — probably, the most popular distro. Software Center, `snap` and `apt` will get you assessed with all the tools you need.
- Fedora — another very popular distro. A repo for PyCharm is shipped with the OS!

Ubuntu and Fedora offer different desktop environments. If you want a macOS-like experience with little clutter and sleek designs, choose GNOME-based variants (the default ones). If you are more familiar with Windows and/or want the highest degree of customizability, choose KDE- or XFCE-based versions (Kubuntu, Fedora KDE). If you only care about performance, choose Xfce-based versions (Xubuntu, Fedora Xfce).

Users of macOS should be fine on their own. It is however recommended they install Homebrew for easier package management.

Windows users can also try WSL and use Linux together with Windows with a good editor support (e.g. remote execution and debugging).

4.1.2 Code Editor

LaTeXBuddy is a (relatively big) Python-based project, so editing it with just a Notepad would be silly. We recommend you install a "smart" code editor, like Visual Studio Code, or an IDE, like PyCharm. Or, if you know what you're doing, you can use Vim.

PyCharm

PyCharm offers the best toolchain for a developer, but can be a bit too heavy on CPU and RAM. Community edition will work fine, but for a better support for Web Frameworks (which partly power LaTeXBuddy) it is recommended you use the Professional version. It can be obtained for free if you're a student or a teacher.

4.1.3 Git

Make sure you've got Git installed.

If you're on macOS or Linux, you either already have it installed or can easily install it from a package manager. For macOS use Homebrew, for other Linux repos it can be different; consult Google for "{DISTRO} package manager".

If you use Windows (without WSL), install Git from the official website. Choose "Git Bash" as your shell as it offers a Linux-like experience.

Most of Git's initial settings are okay, however it still needs to be configured. First and foremost, your name and email; execute the following commands:

```
git config --global user.name "Max Mustermann"  # replace with your name
git config --global user.email "m.mustermann@example.de"  # replace with your email
```

You can also set this up on a per-project basis. Navigate to the repository and replace `--global` with `--local` in the commands above. This will update your email and name for the repository you're in.

When a Git conflict comes our way, we want to rebase our changes rather than merge them — this makes the git tree look cleaner. Execute

```
git config --global pull.rebase true
```

Client

For easier work you may want to use a Git GUI client. Luckily, there is a plethora of choices for you! VSCode and PyCharm offer very robust built-in Git editors. Other good choices include, but are not limited to: Fork, GitHub Desktop, GitKraken, Sourcetree, etc.

4.1.4 Python

Obviously, a version of Python is required for you to develop LaTeXBuddy.

It is recommended, that the development is done using the latest Python 3 version (as this is written, it's 3.11.1). However, since the app aims to support Python versions down to 3.7, it is also recommended, that you have this version installed as well.

> **Note:** on Ubuntu and other Debian-based distros it takes a long time until the newest Python version arrives to the package manager repositories. It can be, that 3.8 is the newest version you can install. It's okay; however, if you want to have the newest version, use a Python version manager or build the needed version from sources.

To be able to "juggle" around Python versions easily, a Python version manager is recommended. pyenv is pretty much the standard.

Windows users can also install both Python 3.7 and Python 3.11 from .exe installers and set up their editors to use separate versions for separate occasions.

4.1.5 Tox

Tox is the environment manager and runner that we use. It is crucial that you install it. You can install it via `pip`; consult the official installation instructions.

4.1.6 pre-commit

Last but not least, we use pre-commit for Git hook management. It will run all the linting and formatting tools every time you commit, so you won't forget it.

Install pre-commit as described on the website, and install the hooks:

```
pre-commit install
```

4.2 Authoring a change

4.2.1 Log your Changes

To track the changes, we use towncrier. Instead of writing all changes to a big CHANGELOG file, we propose you author the changelog entries as small file snippets.

After you have implemented a change, create a file under `changelog.d/` with the name of `<ISSUE_NUMBER>.<CATEGORY>.md`, where `<ISSUE_NUMBER>` is the number of the issue you're closing with your change, and `<CATEGORY>` is one of: `breaking`, `add`, `change`, `fix`, `remove`, `deprecate`, `internal`. If you don't have an issue number (for example, you directly proposed a pull request), use some unique ID with a + sign prepended.

For example, this is the changelog entry for introducing towncrier, stored in the file `+towncrier.internal.md`:

```
Towncrier is now used to generate the changelog
```

4.3 Releasing a new version

This section describes the process of releasing a new version of LaTeXBuddy. We do not have a release schedule; the releases are usually published whenever we want to. This may be changed later to a model where every push to `master` publishes a new version.

Note: LaTeXBuddy is not being published to PyPI as of now. The only way to get it is from GitLab Package Registry. See *install docs* for more information.

4.3.1 Requirements for a release

The releases are usually started by the maintainers. To kick off the initial release discussion, one has to create a new branch off the `master` branch and name it `release/<VERSION>`. For example: `release/0.5.0`. Then, a merge request with the title "Draft: Prepare release <VERSION>" has to be created.

Upon creating the release, one has to do various checks before the merge request can be un-drafted. This includes:

- running tests under every Python version (is usually done by the CI)
- compiling the CHANGELOG with Towncrier:

```
towncrier build --version <VERSION> --yes
```

- checking newly documentation for correct rendering

After doing all that, the merge request can be un-drafted. It still has to get a review of at least one maintainer before it can be merged.

4.3.2 Releasing

Once the release is checked and everything seems fine, a maintainer can proceed with the release:

1. Merge the request into the `master` branch
2. After the pipeline has completed, create a tag `v<VERSION>` on the latest commit
3. Wait for the release pipeline to publish the package and to create a new release

4.4 GNU Free Documentation License

Version 1.3, 3 November 2008

4.4.1 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

4.4.2 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

4.4.3 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

4.4.4 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4.4.5 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

4.4.6 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

4.4.7 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

4.4.8 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

4.4.9 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

4.4.10 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

4.4.11 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See https://www.gnu.org/licenses/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

4.4.12 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

4.4.13 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

PYTHON MODULE INDEX

l

INDEX

G

H

I

K

L

M

N

O

P

R

S

T

W