LaTeXBuddy

Release 0.4.1

LaTeXBuddy team

Dec 09, 2022

CONTENTS

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 0.4.1 - 09 Dec 2022

1.1.1 Fixed

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

1.2 0.4.0 - 09 Dec 2022

1.2.1 BREAKING CHANGES

- minimal Python version set to 3.7 (!168)

1.2.2 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)

1.2.3 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)

1.2.4 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)

1.3 0.3.0 - 15 Jun 2021

1.3.1 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)

1.3.2 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)

1.3.3 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)

1.4 0.2.0 - 08 Jun 2021

1.4.1 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)

1.4.2 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)

1.4.3 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)

1.5 0.1.0 - 18 May 2021

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

1.5.1 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

LaTeXBuddy is temporarily copyrighted. You can expect it to become open-source later (we're thinking of GPL). You can follow the respective issue

2.1 Copyright notices

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

The LaTeXBuddy logo is based on the "Origami" icon, made by Freepik from www.flaticon.com.

CHAPTER

THREE

USER'S GUIDE

3.1 Install

LaTeXBuddy is a Python package and thus can be installed with `pip`

3.1.1 GITZ registry

Attention: This section is outdated. It relies on the GITZ server, where we used to host our project at.

Create a token

To install packages from GITZ's GitLab Package Registry you'll need to create a personal token. Go to https://git.rz.tu-bs.de/-/profile/personal_access_tokens and create a personal token with the `read_api` scope. You can give it any name and expiration date you want.

Add a personal access token

Enter the name of your application, and we'll return a unique personal access token.

Name

PyPI registry

Expires at

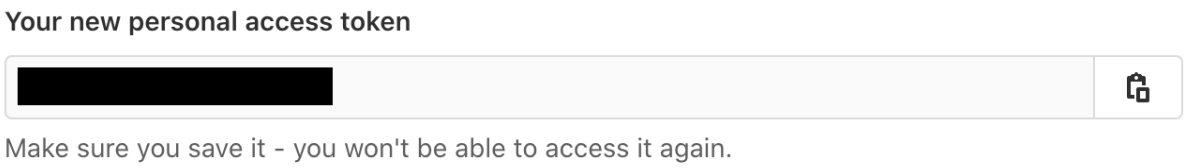
YYYY-MM-DD

Scopes

- [ ] **api**
  Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.
- [ ] **read_user**
  Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.
- [x] **read_api**
  Grants read access to the API, including all groups and projects, the container registry, and the package registry.
- [ ] **read_repository**
  Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
- [ ] **write_repository**
  Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).
- [ ] **read_registry**
  Grants read-only access to container registry images on private projects.
- [ ] **write_registry**
  Grants write access to container registry images on private projects.

Create personal access token

After clicking "Create personal access token", you'll be shown your created token. **This is the only time you'll see it**, so make sure to copy it to not lose it!



Enable VPN

Access to most parts of the GITZ infrastructure is closed for people not affiliated with the university. For good measure, enable your VPN.

Install the package

To install the package, execute the following command:

```
pip install latexbuddy --extra-index-url https://__token__:<your_personal_token>@git.rz.
↪tu-bs.de/api/v4/projects/585/packages/pypi/simple
```

Don't forget to replace `<your_personal_token>` with the token, generated in the previous step!

Other versions

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

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": {
        "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

The one-stop-shop for LaTeX checking.

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

3.5.1 Named Arguments

--version, -V show program's version number and exit

--wl_add_keys, -ak Arguments are valid keys that should be added to whitelist. Ideally the keys are copied from LaTeXBuddy HTML Output

--wl_from_wordlist, -awl First argument is a file containing a single word per line, second argument is the language of the words. The words get parsed to keys and the keys get added to the whitelist as spelling errors that will be ignored by LaTeXBuddy

--flask This option starts a local webserver to check your documents with a GUI.

Default: False

3.6 API Reference

3.6.1 Main instance

This module describes 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

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 – the UID of the error to be deleted

static check_whitelist()

Removes errors that are whitelisted.

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[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[Path]*) – a list of the paths for the html output
- **compile_tex** (*bool*) – boolean if the tex file should be compiled

static output_file()

Writes all current problems to the specified output file.

static output_html()

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

static output_json()

Writes all the current problem objects to the output file.

static run_tools()

Runs all modules in the LaTeXBuddy toolchain in parallel

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 (*Optional[Namespace]*) –

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** – 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** (*Optional[str]*) – regular expression that the config entry is required to match fully (otherwise ConfigOptionVerificationError is raised) Note: this overrides verify_type with 'AnyStr'
- **verify_choices** (*Optional[Union[List[Any], Tuple[Any], Set[Any]]]*) – 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** – 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** (*Optional[str]*) – regular expression that the config entry is required to match fully (otherwise ConfigOptionVerificationError is raised) Note: this overrides verify_type with 'AnyStr'
- **verify_choices** (*Optional[Union[List[Any], Tuple[Any], Set[Any]]]*) – 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
Optional[*Tuple*[int, int]]

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 .py files in the ModuleLoader's directory (or subfolders)

Return type
List[*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[*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[*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[*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 probably 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** (*Optional[Tuple[int, int]]*) –
- **text** (*str*) –
- **file** (*Path*) –
- **severity** (ProblemSeverity) –
- **p_type** (*Optional[str]*) –
- **length** (*Optional[int]*) –
- **category** (*Optional[str]*) –
- **description** (*Optional[str]*) –
- **context** (*Optional[Tuple[str, str]]*) –
- **suggestions** (*Optional[List[str]]*) –
- **key** (*Optional[str]*) –

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*) –

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 wrong LaTeX syntax

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

Sets the static variable language used for key generation

Parameters
lang – global language that the modules currently work with

3.6.6 Output

`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[Interval]]*) – List of lists of Intervals for any given line
- **problems** (*List[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`**(*tex_lines*)

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

Parameters
tex_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*[*Interval*]]

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

Generates and returns a pair of HTML span 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 span 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[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`**(*tex_lines*, *line_intervals*)

Adds HTML marker-tags (span) for every interval in line_intervals to the respective line in tex_lines and escapes all HTML control characters in tex_lines.

Parameters

- **tex_lines** (*List[str]*) – text lines from the .tex-file
- **line_intervals** (*List[List[Interval]]*) – list of non-intersecting intervals to be marked for every line in tex_lines

Return type
None

`latexbuddy.output.`**`mark_intervals_in_tex_line`**(*tex_line*, *intervals*)

Adds HTML marker-tags (span) for every interval in intervals to the respective tex_line string and returns the resulting line. This method also escapes all HTML control characters included in tex_line.

Parameters

- **tex_line** (*str*) – line from the .tex-file
- **intervals** (*List[Interval]*) – list of non-intersecting intervals to be highlighted in the line

Returns
resulting line as a string, containing HTML span 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, 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[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** (*Optional[int]*) –

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 ProblemFilters are:

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

This method explicitly 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 with respect to its type and custom parameters, 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** (*Optional[int]*) –

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 ProblemFilters are:

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

This method explicitly 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 with respect to its type and custom parameters, 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*[Problem]) – list of Problems to filter

Returns
filtered list of Problems

Return type
List[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*)

This method takes in a TexFile object and parses all contained preprocessor commands resulting in a set of ProblemFilters which are added to this instance's list of filters to be applied to any given problem.

Parameters
file (TexFile) – TexFile object containing the LaTeX sourcecode to be parsed

Return type
None

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

Describes the base class for any filter for Problem objects. ProblemFilter provides functionality to define a start and end line and to match a problem based on its line position.

ProblemFilters can be created open-ended by omitting the end_line parameter resulting in the filter matching any Problem located at or below the start line. Open-ended filters can later be closed by supplying an end line to the end() method.

For more diverse filters ProblemFilter provides the following abstract methods which must be implemented by all subclasses:

- **custom_match: Matches a problem based on custom parameters. This method is** only called, if a given problem is located within the filter's line boundaries
- **custom_parameters_equal: determines whether another custom filter is of the** same type and has all equal custom parameters as the current one

Parameters

- **start_line** (*int*) –
- **end_line** (*Optional[int]*) –

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 ProblemFilters are:

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

This method explicitly 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 with respect to its type and custom parameters, 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; false otherwise

Return type
bool

`match`(*problem*)

Determines, whether a given problem is located within the ProblemFilter'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** (*Optional[int]*) –

`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 ProblemFilters are:

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

This method explicitly 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 with respect to its type and custom parameters, False otherwise

Return type
bool

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

WhitelistKeyProblemFilter implementation that filters problems, if they have been created with a specified whitelist key.

Parameters

- **wl_key** (*str*) –
- **start_line** (*int*) –
- **end_line** (*Optional[int]*) –

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 ProblemFilters are:

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

This method explicitly 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 with respect to its type and custom parameters, False otherwise

Return type
bool

3.6.8 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.

Logging

class latexbuddy.log.**ConsoleFormatter**(*enable_colour=True*)

Log formatter for console output.

Outputs messages with colours (thanks to colorama) and severities.

Parameters
enable_colour (*bool*) –

format(*record*)

Format the specified record as text.

The record's attribute dictionary is used as the operand to a string formatting operation which yields the returned string. Before formatting the dictionary, a couple of preparatory steps are carried out. The message attribute of the record is computed using LogRecord.getMessage(). If the formatting string uses the time (as determined by a call to usesTime(), formatTime() is called to format the event time. If there is exception information, it is formatted using formatException() and appended to the message.

Parameters
record (*LogRecord*) –

Return type
str

class latexbuddy.log.**Loggable**

This class provides logging functionality to any class that inherits from it.

property logger

Returns a logger that includes the full module path and the classname in its name.

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.

class latexbuddy.tools.**ToolLogger**

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]]

latexbuddy.tools.**add_whitelist_console**(*whitelist_file*, *to_add*)

Adds a list of keys to the Whitelist. Keys should be valid keys, ideally copied from LaTeXBuddy HTML Output.

Parameters

- **whitelist_file** – Path to whitelist file
- **to_add** – list of keys

latexbuddy.tools.**add_whitelist_from_file**(*whitelist_file*, *file_to_parse*, *lang*)

Takes in a list of words and creates their respective keys, then adds them to whitelist. Words in the file_to_parse should all be from the same language. Each line represents a single Word.

Parameters

- **whitelist_file** – Path to whitelist file
- **file_to_parse** – Path to wordlist
- **lang** – language of the words in the wordlist

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.**convert_file_to_absolute**(*unchecked_files*, *root_dir*)

Converts a relative path to an absolute if needed

Parameters

- **unchecked_files** (*List[Path]*) – the list of unchecked_files
- **root_dir** (*str*) – the root directory

Returns
the absolute path

Return type
Path

`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

`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** (*Optional[str]*) – 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.ToolLogger (WARNING)>*, *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** (*Optional[str]*) – 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_abs_path**(*path*)

Gets absolute path of a string :param path: the path :return: the absolute path

Return type
Path

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[*Path*]

latexbuddy.tools.**get_app_dir**()

Finds the directory for storing application data (mostly logs).

This is a lightweight port of Click's mononymous function: https://github.com/pallets/click/blob/af0af571cbbd921d3974a0ff9cf58a4b26bb852b/src/click/utils.py#L412-L458

Returns
path of the application directory

Return type
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 at least 1024 bytes were read.

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

`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[Path]*) –
- **checked_files** (*List[Path]*) –

Unchecked_files
the unchecked_files

Checked_files
the checked_files

Returns
the unchecked_files

Return type
List[*Path*]

`latexbuddy.tools.`**`perform_whitelist_operations`**(*args*)

Performs whitelist operations

Parameters
`args` (*`Namespace`*) – the args

`latexbuddy.tools.`**`texify_path`**(*path*)

Adds .tex to a file path if needed :param path: the path :return: the texified path

Parameters
`path` (*`str`*) –

Return type
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[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[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[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[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
Optional[*Path*]

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
(<class 'str'>, <class 'str'>, <class '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*, *texfile*)

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** – the file path

Return type
List[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[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[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
Optional[*Dict*]

`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`) – LanguageTool's error output
- **file** (`TexFile`) – TexFile object representing the file to be checked

Return type
List[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
Optional[*Dict*]

`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]*) – list of LT's replacements for a particular word
- **max_elements** (*int*) – Caps the number of 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[Problem]

`class latexbuddy.modules.languagetool.LanguageToolLocalServer`(*logger*)

Defines an instance of a local LanguageTool deployment.

Parameters
logger (*Logger*) –

`static find_free_port`(*port=None*)

Tries to find a free port for the LanguageTool server.

Parameters
port (*int*) – 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[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[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** (*Optional[str]*) –
- **module_selector_mode** (*Optional[str]*) –
- **module_selection** (*Optional[str]*) –
- **whitelist_id** (*Optional[str]*) –

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 Poetry

Poetry is the dependency management tool that we use. It is crucial that you install it. You can install it via `pip`, however it's highly recommended that you use the provided install scripts.

4.1.6 pre-commit

Last but not least, we use pre-commit for Git hook management. It will run all the extra tools, like `isort`, `black`, etc. 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
```

PYTHON MODULE INDEX

l

INDEX

F

G

H

I

K

L

M

N

O

P

R

S

T

W