ConfigUpdater Documentation

Release 3.2

Florian Wilhelm

Aug 19, 2026

CONTENTS

1 Contents **3**

1.1 Usage . . . 3
1.2 Contributing . . . 6
1.3 License . . . 7
1.4 Contributors . . . 11
1.5 Changelog . . . 11
1.6 API Reference . . . 13

2 Indices and tables **29**

Python Module Index **31**

Index **33**



The sole purpose of ConfigUpdater is to easily update an INI config file with no changes to the original file except the intended ones. This means comments, the ordering of sections and key/value-pairs as wells as their cases are kept as in the original file. Thus ConfigUpdater provides complementary functionality to Python's ConfigParser which is primarily meant for reading config files and writing *new* ones. Read more on how to use ConfigUpdater in the *usage page*.

The key differences to ConfigParser are:

- minimal invasive changes in the update configuration file,
- proper handling of comments,
- only a single config file can be updated at a time,
- the original case of sections and keys are kept,
- control over the position of a new section/key

Following features are **deliberately not** implemented:

- interpolation of values,
- propagation of parameters from the default section,
- conversions of values,
- passing key/value-pairs with `default` argument,
- non-strict mode allowing duplicate sections and keys.

Note

ConfigUpdater is mainly developed for PyScaffold.

CHAPTER

ONE

CONTENTS

1.1 Usage

First install the package with:

```
pip install configupdater
```

Now we can simply do:

```
from configupdater import ConfigUpdater

updater = ConfigUpdater()
updater.read("setup.cfg")
```

which would read the file `setup.cfg` that is found in many projects.

To change the value of an existing key we can simply do:

```
updater["metadata"]["author"].value = "Alan Turing"
```

At any point we can print the current state of the configuration file with:

```
print(updater)
```

To update the read-in file just call `updater.update_file()` or `updater.write("filename")` to write the changed configuration file to another destination. Before actually writing, ConfigUpdater will automatically check that the updated configuration file is still valid by parsing it with the help of ConfigParser.

Many of ConfigParser's methods still exists and it's best to look them up in the *API reference*. Let's look at some examples.

1.1.1 Adding and removing options

Let's say we have the following configuration in a string:

```
cfg = """
[metadata]
author = Ada Lovelace
summary = The Analytical Engine
"""
```

We can add an *license* option, i.e. a key/value pair, in the same way we would do with ConfigParser:

```
updater = ConfigUpdater()
updater.read_string(cfg)
updater["metadata"]["license"] = "MIT"
```

A simple `print(updater)` will give show you that the new option was appended to the end:

```
[metadata]
author = Ada Lovelace
summary = The Analytical Engine
license = MIT
```

Since the license is really important to us let's say we want to add it before the `summary` and even add a short comment before it:

```
updater = ConfigUpdater()
updater.read_string(cfg)
(updater["metadata"]["summary"].add_before
                               .comment("Ada would have loved MIT")
                               .option("license", "MIT"))
```

which would result in:

```
[metadata]
author = Ada Lovelace
# Ada would have loved MIT
license = MIT
summary = Analytical Engine calculating the Bernoulli numbers
```

Using `add_after` would give the same result and looks like:

```
updater = ConfigUpdater()
updater.read_string(cfg)
(updater["metadata"]["author"].add_after
                              .comment("Ada would have loved MIT")
                              .option("license", "MIT"))
```

Let's say we want to rename `summary` to the more common `description`:

```
updater = ConfigUpdater()
updater.read_string(cfg)
updater["metadata"]["summary"].key = "description"
```

If we wanted no summary at all, we could just do `del updater["metadata"]["summary"]`.

1.1.2 Adding and removing sections

Adding and remove sections just works like adding and removing options but on a higher level. Sticking to our *Ada Lovelace* example, let's say we want to add a section `options` just before `metadata` with a comment and two new lines to separate it from `metadata`:

```
updater = ConfigUpdater()
updater.read_string(cfg)
(updater["metadata"].add_before
                    .comment("Some specific project options")
```

(continues on next page)

(continued from previous page)

```
                    .section("options")
                    .space(2))
```

As expected, this results in:

```
# Some specific project options
[options]

[metadata]
author = Ada Lovelace
summary = The Analytical Engine
```

We could now fill the new section with options like we learnt before. If we wanted to rename an existing section we could do this with the help of the `name` attribute:

```
updater["metadata"].name = "MetaData"
```

Sometimes it might be useful to inject a new section not in a programmatic way but more declarative. Let's assume we have thus defined our new section in a multi-line string:

```
sphinx_sect_str = """
[build_sphinx]
source_dir = docs
build_dir = docs/_build
"""
```

With the help of two ConfigUpdater objects we can easily inject this section into our example:

```
sphinx = ConfigUpdater()
sphinx.read_string(sphinx_sect_str)
sphinx_sect = sphinx["build_sphinx"]

updater = ConfigUpdater()
updater.read_string(cfg)

(updater["metadata"].add_after
                    .space()
                    .section(sphinx_sect.detach()))
```

The `detach()` method will remove the `build_sphinx` section from the first object and add it to the second object. This results in:

```
[metadata]
author = Ada Lovelace
summary = The Analytical Engine

[build_sphinx]
source_dir = docs
build_dir = docs/_build
```

Alternatively, if you want to preserve `build_sphinx` in both *ConfigUpdater* objects (i.e., prevent it from being removed from the first while still adding a copy to the second), you call also rely on stdlib's `copy.deepcopy()` function instead of `detach()`:

```
from copy import deepcopy

(updater["metadata"].add_after
                    .space()
                    .section(deepcopy(sphinx_sect)))
```

This technique can be used for all objects inside ConfigUpdater: sections, options, comments and blank spaces.

Shallow copies are discouraged in the context of ConfigUpdater because each configuration block keeps a reference to its container to allow easy document editing. When doing editions (such as adding or changing options and comments) based on a shallow copy, the results can be unreliable and unexpected.

For more examples on how the API of ConfigUpdater works it's best to take a look into the unit tests and read the references.

1.2 Contributing

ConfigUpdater is an open-source project and needs your help to improve. If you experience bugs or in general issues, please file an issue report on our issue tracker. If you also want to contribute code or improve the documentation it's best to create a Pull Request (PR) on Github. Here is a short introduction how it works.

1.2.1 Code Contributions

Submit an issue

Before you work on any non-trivial code contribution it's best to first create an issue report to start a discussion on the subject. This often provides additional considerations and avoids unnecessary work.

Create an environment

Before you start coding we recommend to install Miniconda which allows to setup a dedicated development environment named `configupdater` with:

```
conda create -n configupdater python=3 virtualenv pytest pytest-cov
```

Then activate the environment `configupdater` with:

```
source activate configupdater
```

Clone the repository

1. Create a Gitub account if you do not already have one.
2. Fork the project repository: click on the *Fork* button near the top of the page. This creates a copy of the code under your account on the GitHub server.
3. Clone this copy to your local disk:

   ```
   git clone git@github.com:YourLogin/configupdater.git
   ```

4. Run `python setup.py develop` to install `configupdater` into your environment.
5. Install `pre-commit`:

   ```
   pip install pre-commit
   pre-commit install
   ```

PyScaffold project comes with a lot of hooks configured to automatically help the developer to check the code being written.

6. Create a branch to hold your changes:

```
git checkout -b my-feature
```

and start making changes. Never work on the main branch!

7. Start your work on this branch. When you're done editing, do:

```
git add modified_files
git commit
```

to record your changes in Git, then push them to GitHub with:

```
git push -u origin my-feature
```

8. Please check that your changes don't break any unit tests with:

```
python setup.py test
```

Don't forget to also add unit tests in case your contribution adds an additional feature and is not just a bugfix.

9. Use flake8 to check your code style.
10. Add yourself to the list of contributors in `AUTHORS.rst`.
11. Go to the web page of your ConfigUpdater fork, and click "Create pull request" to send your changes to the maintainers for review. Find more detailed information creating a PR.

1.3 License

ConfigUpdater is licensed under the MIT license; see below for details.

ConfigUpdater includes code derived from the Python standard library, which is licensed under the Python license, a permissive open source license. The copyright and license are included at the bottom of this file for compliance with Python's terms.

The MIT License (MIT)

Copyright (c) 2018 Florian Wilhelm

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright (c) 2001-present Python Software Foundation; All Rights Reserved

1.3.1 A. HISTORY OF THE SOFTWARE

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see http://www.cwi.nl) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see http://www.cnri.reston.va.us) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations, which became Zope Corporation. In 2001, the Python Software Foundation (PSF, see https://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation was a sponsoring member of the PSF.

All Python releases are Open Source (see http://www.opensource.org for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Release Derived Year Owner GPL-
from compatible? (1)

0.9.0 thru 1.2 1991-1995 CWI yes 1.3 thru 1.5.2 1.2 1995-1999 CNRI yes 1.6 1.5.2 2000 CNRI no 2.0 1.6 2000 BeOpen.com no 1.6.1 1.6 2001 CNRI yes (2) 2.1 2.0+1.6.1 2001 PSF no 2.0.1 2.0+1.6.1 2001 PSF yes 2.1.1 2.1+2.0.1 2001 PSF yes 2.1.2 2.1.1 2002 PSF yes 2.1.3 2.1.2 2002 PSF yes 2.2 and above 2.1.1 2001-now PSF yes

Footnotes:

(1) GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.

(2) According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is "not incompatible" with the GPL.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

1.3.2 B. TERMS AND CONDITIONS FOR ACCESSING OR OTHERWISE USING PYTHON

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020 Python Software Foundation; All Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python.

4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at http://www.pythonlabs.com/logos.html may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright (c) 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: http://hdl.handle.net/1895.22/1013".

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

1.4 Contributors

- Florian Wilhelm <florian.wilhelm@gmail.com>
- Christian Theune <ct@flyingcircus.io>
- Anderson Bravalheri <andersonbravalheri@gmail.com>

1.5 Changelog

1.5.1 Version 3.2

- Option `prepend_newline` in `set_values` to optionally avoid new lines, issue #104
- Fix square brackets not parsed in section names, issue #122

1.5.2 Version 3.1.1

- Preserve indentation of section when there are comments, issue #92

1.5.3 Version 3.1

- Prevent modifying multi-line values directly with `value`, issue #87
- Added `append` method to `Option` for editing multi-line values
- Added `as_list` method to `Option` to handle multi-line values more easily

1.5.4 Version 3.0.1

- Fix error when parsing unindented comments in multi-line values, issue #73
- Fix invalid string produced when `allow_no_value = False`, issue #68

1.5.5 Version 3.0

- Added type hinting, issue #16
- Fix parsing error of indented comment lines, issue #25
- Allow handling of raw section comment, issue #25
- More unit testing of optionxform, issue #55
- Allowing sections/options to be copied from one document to the other, issue #47
- New logo, issue #29
- Whole API was rechecked by @abravalheri and changed for consistency, issue #19

1.5.6 Version 2.0

- Changes in parser, i.e. comments in multi-line option values are kept
- Issue #14 is fixed
- Parameter `empty_lines_in_values` is now activated by default and can be changed
- Renamed `sections_blocks` to `section_blocks` for consistency
- Renamed `last_item` to `last_block` for consistency
- Added `first_block`
- Reworked some internal parts of the inheritance hierarchy
- Added `remove` to remove the current block
- Added `next_block` and `previous_block` for easier navigation in section

1.5.7 Version 1.1.3

- Added fallback option to ConfigUpdater.get reflecting ConfigParser

1.5.8 Version 1.1.2

- Fix wrongly modifying input in Option.set_value #11

1.5.9 Version 1.1.1

- Fix iterating over the items() view of a section breaks #8

1.5.10 Version 1.1

- Validate format on write by default (can be deactivated)
- Fixed issue #7 with mixed-case options
- Fixed issue #7 with add_before/add_after problem
- Fixed issue #7 with wrong duplicate mixed-case entries
- Fixed issue #7 with duplicate options after add_after/before

1.5.11 Version 1.0.1

- More sane error message if `read_file` is misused
- Also run unittests with Windows

1.5.12 Version 1.0

- Fix: Use n instead of os.linesep where appropriate

1.5.13 Version 0.3.2

- Added Github and documentation link into the project's metadata

1.5.14 Version 0.3.1

- Require Python >= 3.4 with `python_requires`

1.5.15 Version 0.3

- Added a `insert_at` function at section level
- Some internal code simplifications

1.5.16 Version 0.2

- Added a `to_dict()` function

1.5.17 Version 0.1.1

- Allow for flexible comment character in `comment(...)`

1.5.18 Version 0.1

- First release

1.6 API Reference

exception configupdater.**AlreadyAttachedError**(*block: str | Block = 'The block'*)

Bases: `Exception`

{block} has been already attached to a container.

Try to remove it first using `detach` or create a copy using stdlib's `copy.deepcopy`.

add_note(*note*, */ (Positional-only parameter separator (PEP 570))*)

Add a note to the exception

with_traceback(*tb*, */*)

Set self.__traceback__ to tb and return self.

exception configupdater.**AssignMultilineValueError**(*block: str | Block = 'The block'*)

Bases: `Exception`

Trying to assign a multi-line value to {block}. Use the `set_values` or `append` method to accomplish that.

add_note(*note*, */*)

Add a note to the exception

with_traceback(*tb*, */*)

Set self.__traceback__ to tb and return self.

class configupdater.**Comment**(*container: Container | None = None*)

Bases: `Block`

Comment block

property add_after: BlockBuilder

Block builder inserting a new block after the current block

property add_before: BlockBuilder

Block builder inserting a new block before the current block

add_line(*line: str*) → B

PRIVATE: this function is not part of the public API of Block. It is only used internally by other classes of the package during parsing.

Add a line to the current block

Parameters
line (*str*) – one line to add

attach(*container: Container*) → B

PRIVATE: Don't use this as a user!

Rather use *add_** or the bracket notation

property container: Container

Container holding the block

property container_idx: int

Index of the block within its container

detach() → B

Remove and return this block from container

has_container() → bool

Checks if this block has a container attached

property next_block: Block | None

Next block in the current container

property previous_block: Block | None

Previous block in the current container

property updated: bool

True if the option was changed/updated, otherwise False

class configupdater.**ConfigUpdater**(*allow_no_value=False*, ** (Keyword-only parameters separator (PEP 3102))*, *delimiters: Tuple[str, ...] = ('=', ':')*, *comment_prefixes: Tuple[str, ...] = ('#', ';')*, *inline_comment_prefixes: Tuple[str, ...] | None = None*, *strict: bool = True*, *empty_lines_in_values: bool = True*, *space_around_delimiters: bool = True*)

Bases: `Document`

Tool to parse and modify existing `cfg` files.

ConfigUpdater follows the API of ConfigParser with some differences:

- inline comments are treated as part of a key's value,
- only a single config file can be updated at a time,
- the original case of sections and keys are kept,
- control over the position of a new section/key.

Following features are **deliberately not** implemented:

- interpolation of values,
- propagation of parameters from the default section,
- conversions of values,
- passing key/value-pairs with `default` argument,

- non-strict mode allowing duplicate sections and keys.

ConfigUpdater objects can be created by passing the same kind of arguments accepted by the *Parser*. After a ConfigUpdater object is created, you can load some content into it by calling any of the `read*` methods (*read()*, *read_file()* and *read_string()*).

Once the content is loaded you can use the ConfigUpdater object more or less in the same way you would use a nested dictionary. Please have a look into `Document` to understand the main differences.

When you are done changing the configuration file, you can call *write()* or *update_file()* methods.

add_section(*section: str | Section*)

Create a new section in the configuration.

Raise DuplicateSectionError if a section by the specified name already exists. Raise ValueError if name is DEFAULT.

Parameters
section (str or *Section*) – name or Section type

clear() → None. Remove all items from D.

get(*section*, *option*, *fallback=UniqueValues._UNSET*)

Gets an option object for a given section or a fallback value.

Warning

Please notice this method works differently from what is expected of `MutableMapping.get()` (or `dict.get()`). Similarly to `configparser.ConfigParser.get()`, will take least 2 arguments, and the second argument does not correspond to a default value.

This happens because this function is not designed to return a *Section* of the *ConfigUpdater* document, but instead a nested *Option*.

See *get_section()*, if instead, you want to retrieve a *Section*.

Parameters

- **section** (*str*) – section name
- **option** (*str*) – option name
- **fallback** (*T*) – if the key is not found and fallback is provided, the `fallback` value will be returned. `None` is a valid fallback value.

Attention

When `option` is not present, the `fallback` value itself is returned. If you want instead to obtain a new `Option` object with a default value associated with it, you can try the following:

```
configupdater.get("section", "option", fallback=Option("name", value))
```

... which roughly corresponds to:

```
configupdater["section"].get("option", Option("name", value))
```

Raises

- *NoSectionError* – if `section` cannot be found
- *NoOptionError* – if the option cannot be found and no `fallback` was given

Returns
Option object holding key/value pair when it exists. Otherwise, the value passed via the `fallback` argument itself (type `T`).

`get_section`(*name*, *default=None*)

This method works similarly to `dict.get()`, and allows you to retrieve an entire section by its name, or provide a `default` value in case it cannot be found.

`has_option`(*section: str*, *option: str*) → bool

Checks for the existence of a given option in a given section.

Parameters
- **section** (*str*) – name of section
- **option** (*str*) – name of option

Returns
whether the option exists in the given section

Return type
bool

`has_section`(*key*) → bool

Returns whether the given section exists.

Parameters
key (*str*) – name of section

Returns
wether the section exists

Return type
bool

`items`(*section=UniqueValues._UNSET*)

Return a list of (name, value) tuples for options or sections.

If section is given, return a list of tuples with (name, value) for each option in the section. Otherwise, return a list of tuples with (section_name, section_type) for each section.

Parameters
section (*str*) – optional section name, default UNSET

Returns
list of *Section* or *Option* objects

Return type
list

`iter_blocks`() → Iterator[T]

Iterate over all blocks inside container.

`iter_sections`() → Iterator[*Section*]

Iterate only over section blocks

`keys`() → a set-like object providing a view on D's keys

options(*section: str*) → list[str]

Returns list of configuration options for the named section.

Parameters
section (*str*) – name of section

Returns
list of option names

Return type
list

optionxform(*optionstr*) → str

Converts an option key for unification

By default it uses `str.lower()`, which means that ConfigUpdater will compare options in a case insensitive way.

This implementation mimics ConfigParser API, and can be configured as described in `configparser.ConfigParser.optionxform()`.

Parameters
optionstr (*str*) – key name

Returns
unified option name

Return type
str

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise KeyError if D is empty.

read(*filename: str | bytes | PathLike*, *encoding: str | None = None*) → T

Read and parse a filename.

Parameters

- **filename** (*str*) – path to file
- **encoding** (*str*) – encoding of file, default None

read_file(*f: Iterable[str]*, *source: str | None = None*) → T

Like read() but the argument must be a file-like object.

The `f` argument must be iterable, returning one line at a time. Optional second argument is the `source` specifying the name of the file being read. If not given, it is taken from f.name. If `f` has no `name` attribute, <???> is used.

Parameters

- **f** – file like object
- **source** (*str*) – reference name for file object, default None

read_string(*string: str*, *source='<string>'*) → T

Read configuration from a given string.

Parameters

- **string** (*str*) – string containing a configuration

- **source** (*str*) – reference name for file object, default '<string>'

remove_option(*section: str*, *option: str*) → bool

Remove an option.

Parameters

- **section** (*str*) – section name
- **option** (*str*) – option name

Returns
whether the option was actually removed

Return type
bool

remove_section(*name: str*) → bool

Remove a file section.

Parameters
name – name of the section

Returns
whether the section was actually removed

Return type
bool

section_blocks() → list[*Section*]

Returns all section blocks

Returns
list of *Section* blocks

Return type
list

sections() → list[str]

Return a list of section names

Returns
list of section names

Return type
list

set(*section: str*, *option: str*, *value: None | str | Iterable[str] = None*) → D

Set an option.

Parameters

- **section** – section name
- **option** – option name
- **value** – value, default None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict() → dict[str, dict[str, str | None]]

Transform to dictionary

Returns
dictionary with same content

Return type
dict

update([*E*,]**F) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E.keys(): D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

update_file(*validate: bool = True*) → T

Update the read-in configuration file.

Parameters
validate (*Boolean*) – validate format before writing

validate_format(***kwargs*)

Given the current state of the `ConfigUpdater` object (e.g. after modifications), validate its INI/CFG textual representation by parsing it with `ConfigParser`.

The ConfigParser object is instead with the same arguments as the original ConfigUpdater object, but the `kwargs` can be used to overwrite them.

See `validate_format()`.

values() → an object providing a view on D's values

write(*fp: TextIO*, *validate: bool = True*)

Write an .cfg/.ini-format representation of the configuration state.

Parameters

- **fp** (*file-like object*) – open file handle
- **validate** (*Boolean*) – validate format before writing

exception configupdater.**DuplicateOptionError**(*section*, *option*, *source=None*, *lineno=None*)

Bases: `Error`

Raised by strict parsers when an option is repeated in an input source.

Current implementation raises this exception only when an option is found more than once in a single file, string or dictionary.

add_note(*note*, */*)

Add a note to the exception

with_traceback(*tb*, */*)

Set self.__traceback__ to tb and return self.

exception configupdater.**DuplicateSectionError**(*section*, *source=None*, *lineno=None*)

Bases: `Error`

Raised when a section is repeated in an input source.

Possible repetitions that raise this exception are: multiple creation using the API or in strict parsers when a section is found more than once in a single input file, string or dictionary.

add_note(*note*, */*)

Add a note to the exception

with_traceback(*tb*, /)

Set self.__traceback__ to tb and return self.

exception configupdater.**InconsistentStateError**(*msg*, *fpname='<???>'*, *lineno: int = -1*, *line: str = '???'*)

Bases: Exception

Internal parser error, some of the parsing algorithm assumptions was violated, and the internal state machine ended up in an unpredicted state.

add_note(*note*, /)

Add a note to the exception

with_traceback(*tb*, /)

Set self.__traceback__ to tb and return self.

exception configupdater.**MissingSectionHeaderError**(*filename*, *lineno*, *line*)

Bases: *ParsingError*

Raised when a key-value pair is found before any section header.

add_note(*note*, /)

Add a note to the exception

with_traceback(*tb*, /)

Set self.__traceback__ to tb and return self.

exception configupdater.**NoConfigFileReadError**

Bases: Error

Raised when no configuration file was read but update requested.

add_note(*note*, /)

Add a note to the exception

with_traceback(*tb*, /)

Set self.__traceback__ to tb and return self.

exception configupdater.**NoOptionError**(*option*, *section*)

Bases: Error

A requested option was not found.

add_note(*note*, /)

Add a note to the exception

with_traceback(*tb*, /)

Set self.__traceback__ to tb and return self.

exception configupdater.**NoSectionError**(*section*)

Bases: Error

Raised when no section matches a requested option.

add_note(*note*, /)

Add a note to the exception

with_traceback(*tb*, /)

Set self.__traceback__ to tb and return self.

exception configupdater.**NoneValueDisallowed**

Bases: SyntaxWarning

Cannot represent <{option} = None>, it will be converted to <{option} = ''>. Please use allow_no_value=True with ConfigUpdater.

add_note(*note*, /)

Add a note to the exception

with_traceback(*tb*, /)

Set self.__traceback__ to tb and return self.

exception configupdater.**NotAttachedError**(*block: str | Block = 'The block'*)

Bases: Exception

{block} is not attached to a container yet. Try to insert it first.

add_note(*note*, /)

Add a note to the exception

with_traceback(*tb*, /)

Set self.__traceback__ to tb and return self.

class configupdater.**Option**(*key: str*, *value: str | None = None*, *container:* Section *| None = None*, *delimiter: str = '='*, *space_around_delimiters: bool = True*, *line: str | None = None*)

Bases: Block

Option block holding a key/value pair.

property add_after: BlockBuilder

Block builder inserting a new block after the current block

property add_before: BlockBuilder

Block builder inserting a new block before the current block

add_line(*line: str*)

PRIVATE: this function is not part of the public API of Option. It is only used internally by other classes of the package during parsing.

add_value(*value: str | None*)

PRIVATE: this function is not part of the public API of Option. It is only used internally by other classes of the package during parsing.

append(*value: str*, ***kwargs*) → *Option*

Append a value to a mult-line value

Parameters

- **value** (*str*) – value
- **kwargs** – keyword arguments for *set_values*

as_list(*separator='\n'*) → list[str]

Returns the (multi-line/element) value as a list

Empty list if value is None, single-element list for a one-element value and an element for each line in a multi-element value.

Parameters

separator (*str*) – separator for values, default: line separator

attach(*container: Container*) → B

PRIVATE: Don't use this as a user!

Rather use *add_** or the bracket notation

property container: Container

Container holding the block

property container_idx: int

Index of the block within its container

detach() → B

Remove and return this block from container

has_container() → bool

Checks if this block has a container attached

property key: str

Key string associated with the option.

Please notice that the option key is normalized with `optionxform()`.

When the option object is `detached`, this method will raise a *NotAttachedError*.

property next_block: Block | None

Next block in the current container

property previous_block: Block | None

Previous block in the current container

property raw_key: str

Equivalent to *key*, but before applying `optionxform()`.

set_values(*values: Iterable[str]*, *separator='\n'*, *indent: str | None = None*, *prepend_newline=True*)

Sets the value to a given list of options, e.g. multi-line values

Parameters

- **values** (*iterable*) – sequence of values
- **separator** (*str*) – separator for values, default: line separator
- **indent** (*optional str*) – indentation in case of line separator. If prepend_newline is *True* 4 whitespaces by default, otherwise determine automatically if *None*.
- **prepend_newline** (*bool*) – start with a new line or not, resp.

property updated: bool

True if the option was changed/updated, otherwise False

property value: str | None

Value associated with the given option.

value_start_idx() → int

Index where the value of the option starts, good for indentation

class configupdater.**Parser**(*allow_no_value=False, *, delimiters: ~typing.Tuple[str, ...] = ('=', ':'), comment_prefixes: ~typing.Tuple[str, ...] = ('#', ';'), inline_comment_prefixes: ~typing.Tuple[str, ...] | None = None, strict: bool = True, empty_lines_in_values: bool = True, space_around_delimiters: bool = True, optionxform: ~typing.Callable[[str], str] = <class 'str'>*)

Bases: `object`

Parser for updating configuration files.

ConfigUpdater's parser follows ConfigParser with some differences:

- inline comments are treated as part of a key's value,
- only a single config file can be updated at a time,
- the original case of sections and keys are kept,
- control over the position of a new section/key.

Following features are **deliberately not** implemented:

- interpolation of values,
- propagation of parameters from the default section,
- conversions of values,
- passing key/value-pairs with `default` argument,
- non-strict mode allowing duplicate sections and keys.

read(*filename*, *encoding=None*, *into=None*)

Read and parse a filename.

Parameters

- **filename** (*str*) – path to file
- **encoding** (*Optional[str]*) – encoding of file, default None
- **into** (*Optional[Document]*) – object to be populated with the parsed config

read_file(*f*, *source=None*, *into=None*)

Like read() but the argument must be a file-like object.

The `f` argument must be iterable, returning one line at a time. Optional second argument is the `source` specifying the name of the file being read. If not given, it is taken from f.name. If `f` has no `name` attribute, <???> is used.

Parameters

- **f** – file like object
- **source** (*Optional[str]*) – reference name for file object, default None
- **into** (*Optional[Document]*) – object to be populated with the parsed config

read_string(*string*, *source='<string>'*, *into=None*)

Read configuration from a given string.

Parameters

- **string** (*str*) – string containing a configuration
- **source** (*str*) – reference name for file object, default '<string>'
- **into** (*Optional[Document]*) – object to be populated with the parsed config

exception configupdater.**ParsingError**(*source*, **args*)

Bases: Error

Raised when a configuration file does not follow legal syntax.

add_note(*note*, /)

Add a note to the exception

with_traceback(*tb*, /)

Set self.__traceback__ to tb and return self.

class configupdater.**Section**(*name: str*, *container: Document | None = None*, *raw_comment: str = ''*)

Bases: Block, Container[*Option* | *Comment* | *Space*], MutableMapping[str, Option]

Section block holding options

property add_after: BlockBuilder

Block builder inserting a new block after the current block

property add_before: BlockBuilder

Block builder inserting a new block before the current block

add_comment(*line: str*) → S

Add a Comment object to the section

Used during initial parsing mainly

Parameters
line (*str*) – one line in the comment

add_line(*line: str*) → B

PRIVATE: this function is not part of the public API of Block. It is only used internally by other classes of the package during parsing.

Add a line to the current block

Parameters
line (*str*) – one line to add

add_option(*entry: Option*) → S

Add an Option object to the section

Used during initial parsing mainly

Parameters
entry (Option) – key value pair as Option object

add_space(*line: str*) → S

Add a Space object to the section

Used during initial parsing mainly

Parameters
line (*str*) – one line that defines the space, maybe whitespaces

attach(*container: Container*) → B

PRIVATE: Don't use this as a user!

Rather use *add_** or the bracket notation

clear() → None. Remove all items from D.

property container: Container

Container holding the block

property container_idx: int

Index of the block within its container

create_option(*key: str*, *value: str | None = None*) → *Option*

Creates an option with kwargs that respect syntax options given to the parent ConfigUpdater object (e.g. `space_around_delimiters`).

> **Warning**
>
> This is a low level API, not intended for public use. Prefer *set()* or `__setitem__()`.

detach() → B

Remove and return this block from container

get(*key*, *default=None*)

This method works similarly to `dict.get()`, and allows you to retrieve an option object by its key.

has_container() → bool

Checks if this block has a container attached

has_option(*key*) → bool

Returns whether the given option exists.

Parameters
option (*str*) – name of option

Returns
whether the section exists

Return type
bool

insert_at(*idx: int*) → BlockBuilder

Returns a builder inserting a new block at the given index

Parameters
idx (*int*) – index where to insert

items() → list[Tuple[str, *Option*]]

Return a list of (name, option) tuples for each option in this section.

Returns
list of (name, *Option*) tuples

Return type
list

iter_blocks() → Iterator[T]

Iterate over all blocks inside container.

iter_options() → Iterator[*Option*]

Iterate only over option blocks

keys() → a set-like object providing a view on D's keys

property name: str

Name of the section

property next_block: Block | None

Next block in the current container

option_blocks() → list[*Option*]

Returns option blocks

Returns
list of *Option* blocks

Return type
list

options() → list[str]

Returns option names

Returns
list of option names as strings

Return type
list

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.

If key is not found, d is returned if given, otherwise KeyError is raised.

popitem() → (k, v), remove and return some (key, value) pair

as a 2-tuple; but raise KeyError if D is empty.

property previous_block: Block | None

Previous block in the current container

property raw_comment

Raw comment (includes comment mark) inline with the section header

set(*option: str*, *value: None | str | Iterable[str] = None*) → S

Set an option for chaining.

Parameters

- **option** – option name
- **value** – value, default None

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

to_dict() → dict[str, str | None]

Transform to dictionary

Returns
dictionary with same content

Return type
dict

update([*E*,]**F) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E.keys(): D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

property updated: bool

True if the option was changed/updated, otherwise False

values() → an object providing a view on D's values

class configupdater.**Space**(*container: Container | None = None*)

Bases: Block

Vertical space block of new lines

property add_after: BlockBuilder

Block builder inserting a new block after the current block

property add_before: BlockBuilder

Block builder inserting a new block before the current block

add_line(*line: str*) → B

PRIVATE: this function is not part of the public API of Block. It is only used internally by other classes of the package during parsing.

Add a line to the current block

Parameters
line (*str*) – one line to add

attach(*container: Container*) → B

PRIVATE: Don't use this as a user!

Rather use *add_** or the bracket notation

property container: Container

Container holding the block

property container_idx: int

Index of the block within its container

detach() → B

Remove and return this block from container

has_container() → bool

Checks if this block has a container attached

property next_block: Block | None

Next block in the current container

property previous_block: Block | None

Previous block in the current container

property updated: bool

True if the option was changed/updated, otherwise False

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

configupdater, 13

INDEX

A

add_after (*configupdater.Comment property*), 13
add_after (*configupdater.Option property*), 21
add_after (*configupdater.Section property*), 24
add_after (*configupdater.Space property*), 27
add_before (*configupdater.Comment property*), 13
add_before (*configupdater.Option property*), 21
add_before (*configupdater.Section property*), 24
add_before (*configupdater.Space property*), 27
add_comment() (*configupdater.Section method*), 24
add_line() (*configupdater.Comment method*), 13
add_line() (*configupdater.Option method*), 21
add_line() (*configupdater.Section method*), 24
add_line() (*configupdater.Space method*), 27
add_note() (*configupdater.AlreadyAttachedError method*), 13
add_note() (*configupdater.AssignMultilineValueError method*), 13
add_note() (*configupdater.DuplicateOptionError method*), 19
add_note() (*configupdater.DuplicateSectionError method*), 19
add_note() (*configupdater.InconsistentStateError method*), 20
add_note() (*configupdater.MissingSectionHeaderError method*), 20
add_note() (*configupdater.NoConfigFileReadError method*), 20
add_note() (*configupdater.NoneValueDisallowed method*), 21
add_note() (*configupdater.NoOptionError method*), 20
add_note() (*configupdater.NoSectionError method*), 20
add_note() (*configupdater.NotAttachedError method*), 21
add_note() (*configupdater.ParsingError method*), 24
add_option() (*configupdater.Section method*), 24
add_section() (*configupdater.ConfigUpdater method*), 15
add_space() (*configupdater.Section method*), 24
add_value() (*configupdater.Option method*), 21
AlreadyAttachedError, 13
append() (*configupdater.Option method*), 21
as_list() (*configupdater.Option method*), 21
AssignMultilineValueError, 13
attach() (*configupdater.Comment method*), 14
attach() (*configupdater.Option method*), 21
attach() (*configupdater.Section method*), 24
attach() (*configupdater.Space method*), 27

C

clear() (*configupdater.ConfigUpdater method*), 15
clear() (*configupdater.Section method*), 24
Comment (*class in configupdater*), 13
configupdater
    module, 13
ConfigUpdater (*class in configupdater*), 14
container (*configupdater.Comment property*), 14
container (*configupdater.Option property*), 22
container (*configupdater.Section property*), 24
container (*configupdater.Space property*), 27
container_idx (*configupdater.Comment property*), 14
container_idx (*configupdater.Option property*), 22
container_idx (*configupdater.Section property*), 25
container_idx (*configupdater.Space property*), 27
create_option() (*configupdater.Section method*), 25

D

detach() (*configupdater.Comment method*), 14
detach() (*configupdater.Option method*), 22
detach() (*configupdater.Section method*), 25
detach() (*configupdater.Space method*), 27
DuplicateOptionError, 19
DuplicateSectionError, 19

G

get() (*configupdater.ConfigUpdater method*), 15
get() (*configupdater.Section method*), 25
get_section() (*configupdater.ConfigUpdater method*), 16

H

has_container() (*configupdater.Comment method*), 14
has_container() (*configupdater.Option method*), 22
has_container() (*configupdater.Section method*), 25

has_container() (*configupdater.Space method*), 27
has_option() (*configupdater.ConfigUpdater method*), 16
has_option() (*configupdater.Section method*), 25
has_section() (*configupdater.ConfigUpdater method*), 16

I

InconsistentStateError, 20
insert_at() (*configupdater.Section method*), 25
items() (*configupdater.ConfigUpdater method*), 16
items() (*configupdater.Section method*), 25
iter_blocks() (*configupdater.ConfigUpdater method*), 16
iter_blocks() (*configupdater.Section method*), 25
iter_options() (*configupdater.Section method*), 25
iter_sections() (*configupdater.ConfigUpdater method*), 16

K

key (*configupdater.Option property*), 22
keys() (*configupdater.ConfigUpdater method*), 16
keys() (*configupdater.Section method*), 25

M

MissingSectionHeaderError, 20
module
    configupdater, 13

N

name (*configupdater.Section property*), 25
next_block (*configupdater.Comment property*), 14
next_block (*configupdater.Option property*), 22
next_block (*configupdater.Section property*), 26
next_block (*configupdater.Space property*), 27
NoConfigFileReadError, 20
NoneValueDisallowed, 20
NoOptionError, 20
NoSectionError, 20
NotAttachedError, 21

O

Option (*class in configupdater*), 21
option_blocks() (*configupdater.Section method*), 26
options() (*configupdater.ConfigUpdater method*), 16
options() (*configupdater.Section method*), 26
optionxform() (*configupdater.ConfigUpdater method*), 17

P

Parser (*class in configupdater*), 22
ParsingError, 23
pop() (*configupdater.ConfigUpdater method*), 17
pop() (*configupdater.Section method*), 26
popitem() (*configupdater.ConfigUpdater method*), 17
popitem() (*configupdater.Section method*), 26
previous_block (*configupdater.Comment property*), 14
previous_block (*configupdater.Option property*), 22
previous_block (*configupdater.Section property*), 26
previous_block (*configupdater.Space property*), 27

R

raw_comment (*configupdater.Section property*), 26
raw_key (*configupdater.Option property*), 22
read() (*configupdater.ConfigUpdater method*), 17
read() (*configupdater.Parser method*), 23
read_file() (*configupdater.ConfigUpdater method*), 17
read_file() (*configupdater.Parser method*), 23
read_string() (*configupdater.ConfigUpdater method*), 17
read_string() (*configupdater.Parser method*), 23
remove_option() (*configupdater.ConfigUpdater method*), 18
remove_section() (*configupdater.ConfigUpdater method*), 18

S

Section (*class in configupdater*), 24
section_blocks() (*configupdater.ConfigUpdater method*), 18
sections() (*configupdater.ConfigUpdater method*), 18
set() (*configupdater.ConfigUpdater method*), 18
set() (*configupdater.Section method*), 26
set_values() (*configupdater.Option method*), 22
setdefault() (*configupdater.ConfigUpdater method*), 18
setdefault() (*configupdater.Section method*), 26
Space (*class in configupdater*), 27

T

to_dict() (*configupdater.ConfigUpdater method*), 18
to_dict() (*configupdater.Section method*), 26

U

update() (*configupdater.ConfigUpdater method*), 19
update() (*configupdater.Section method*), 26
update_file() (*configupdater.ConfigUpdater method*), 19
updated (*configupdater.Comment property*), 14
updated (*configupdater.Option property*), 22
updated (*configupdater.Section property*), 26
updated (*configupdater.Space property*), 27

V

validate_format() (*configupdater.ConfigUpdater method*), 19

value (*configupdater.Option property*), 22
value_start_idx() (*configupdater.Option method*), 22
values() (*configupdater.ConfigUpdater method*), 19
values() (*configupdater.Section method*), 27

W

with_traceback() (*configupdater.AlreadyAttachedError method*), 13
with_traceback() (*configupdater.AssignMultilineValueError method*), 13
with_traceback() (*configupdater.DuplicateOptionError method*), 19
with_traceback() (*configupdater.DuplicateSectionError method*), 19
with_traceback() (*configupdater.InconsistentStateError method*), 20
with_traceback() (*configupdater.MissingSectionHeaderError method*), 20
with_traceback() (*configupdater.NoConfigFileReadError method*), 20
with_traceback() (*configupdater.NoneValueDisallowed method*), 21
with_traceback() (*configupdater.NoOptionError method*), 20
with_traceback() (*configupdater.NoSectionError method*), 20
with_traceback() (*configupdater.NotAttachedError method*), 21
with_traceback() (*configupdater.ParsingError method*), 24
write() (*configupdater.ConfigUpdater method*), 19