”ConfigMix” Documentation

Version 0.23.3

Franz Glasner

2023-12-07

Last updated on 2026-05-20 (rev dev-46578f03a7cb)

Contents

CHAPTER 1

Short Overview

Author
Franz Glasner

Version
0.23.3

Date
2023-12-07

Revision
dev-46578f03a7cb

“ConfigMix” is a library to help with the configuration of applications and their configuration files.

It reads a couple of configuration files in the following different styles:

- YAML files
- JSON files
- INI files
- TOML files
- executable Python scripts

It then merges the parsed contents of given files into a *unified* tree-ish configuration dictionary and presents them as such to the application. Reading can be done with variable interpolation from other configuration values.

1.1 References and Inspirations

Search for "yaml" on *PyPi*

- https://pypi.python.org/pypi/layered-yaml-attrdict-config/16.1.0

  The package and it's "Links" section

- https://configloader.readthedocs.org/en/latest/

  For the API

CHAPTER 2

Introduction

The configurations can be read from different types of files:

- *YAML files*
- *JSON files*
- *INI files*
- *TOML files*
- *executable Python scripts*

2.1 YAML Files

Need the `yaml` package (https://github.com/yaml/pyyaml) (e.g. `pip install pyyaml`)

Note: All strings are returned as Unicode text strings.

Note: The root object must be a *mapping* and therefore decode into a Python `dict`[1] alike. This is checked by the implementation.

An example is:

```
# -*- coding: utf-8; mode: yaml; indent-tabs-mode: nil; -*-
%YAML 1.1
---

key1: in the root namespace
key2: in the root namespace -- too

tree1:
  key3: 0x20
  tree2:
    key4: get this as 'tree1.tree2.key4'
    key5: true
    key6: 'off'
    key7: []
    key8:
      - val1
      - val2
      - '{{key1}}'
    key9: {}
```

[1] https://docs.python.org/3/library/stdtypes.html#dict

2.2 JSON files

Read the JSON file with the help of Python's native `json`[2] package.

Note: All strings are returned as Unicode text strings.

Note: The root object must be an *object* and therefore decode into a Python `dict`[3] alike. This is checked by the implementation.

An example is:

```
{"key1": "in the root namespace",
 "key2": "in the root namespace -- too",
 "tree1": {
     "key3": 32,
     "tree2": {
         "key4": "get this as 'tree1.tree2.key4'",
         "key5": true,
         "key6": "off",
         "key7": [],
         "key8": [ "val1", "val2", "{{key1}}" ],
         "key9": {}
     }
 }
}
```

For comments in JSON files see section *Comments*.

2.3 INI Files

Read the file and all sections named in parameter *extract* are flattened into the resulting dictionary. By default the section named `config` is used as root section.

Normally all values are returned as Unicode text strings. But values can be annotated and therefore interpreted as other types:

`:int:`
: The value is handled in the same way as a Python `int`[4] literal

`:float:`
: The value is interpreted as `float`[5]

`:bool:`
: The resulting value is a `bool`[6] where

[2] https://docs.python.org/3/library/json.html#module-json
[3] https://docs.python.org/3/library/stdtypes.html#dict
[4] https://docs.python.org/3/library/functions.html#int
[5] https://docs.python.org/3/library/functions.html#float
[6] https://docs.python.org/3/library/functions.html#bool

`1`, `true`, `yes`, `on`
: yield a Python `True`

`0`, `false`, `no`, `off`
: yield a Python `False`

The evaluation is done *case-insensitively*.

Note: All strings are returned as Unicode text strings.

Note: Contrary to the behaviour of the standard Python `configparser`[7] module the INI file reader is *case-sensitive*.

The example INI style configuration below yields an equivalent configuration to the YAML configuration above:

```
# -*- coding: utf-8 -*-

[DEFAULT]
replvalue1 = in the root namespace -- too

[config]
key1 = in the root namespace
key2 = %(replvalue1)s

[config.tree1]
key3 = :int:0x20

[config.tree1.tree2]
key4 = get this as 'tree1.tree2.key4'
key5 = :bool:TRUE
key6 = off

[config.tree1.tree2.key9]
```

As can be seen in this example – INI file internal value interpolation is done as in Python's standard `configparser`[8] module.

This example also illustrates how INI sections are used to build a tree-ish configuration dictionary.

2.4 TOML Files

Read the TOML file with the help of the pure Python `toml` package (https://github.com/uiri/toml) (e.g. `pip install toml`).

All TOML features map seamingless to "ConfigMix".

The example TOML style configuration below yields an equivalent configuration to the YAML configuration above:

[7] https://docs.python.org/3/library/configparser.html#module-configparser
[8] https://docs.python.org/3/library/configparser.html#module-configparser

```
# -*- coding: utf-8 -*-

key1 = "in the root namespace"
key2 = "in the root namespace -- too"

[tree1]
key3 = 0x20

[tree1.tree2]
key4 = "get this as `tree1.tree2.key4'"
key5 = true
key6 = "off"
key7 = []
key8 = [
  "val1",
  "val2",
  "{{key1}}"
]

[tree1.tree2.key9]
```

2.5 Executable Python Scripts

What will be exported:

1. If loading is done with the *extract* parameter only the given keys are extracted from the script.
2. Otherwise it is checked if the scripts defines an `__all__` sequence. If there is one it's contents are the keys to be extracted.
3. If there is no `__all__` object all names not starting with an underscore _ are found.

This is analogous to as Python modules behave when importing them with `from module import *`.

Note: The Python configuration files are evaluated with `exec` and not imported.

The example configuration by Python script below yields an equivalent configuration to the YAML configuration above:

```
# -*- coding: utf-8 -*-

key1 = u"in the root namespace"
key2 = u"in the root namespace -- too"

tree1 = {
    'key3': 0x20,

    'tree2': {
        'key4': u"get this as `tree1.tree2.key4'",
        'key5': True,
        'key6': u"off",
```

(continues on next page)

(continued from previous page)

```
        'key7': [],
        'key8': [
            u"val1",
            u"val2",
            u"{{key1}}"
        ],
        'key9': {}
    }
}
```

2.6 Loading and Merging

Basic usage of the API is as follows in this example:

```
import configmix

#
# Note: With conf10 merging is rather pointless because the tree
# files # are really the same configuration. But is doesn't harm
# also here.
#
config = configmix.load("conf10.yml", "conf10.ini", "conf10.py")

# Get a -- possibly interpolated -- configuration variable's value
value1 = config.getvar_s("key1")

# Get a -- possibly interpolated -- variable from within the tree
value2 = config.getvar_s("tree1.tree2.key4")
```

By default filenames of the configuration files must have the extensions (case-sensitivety depends on your OS):

`.ini`
: for INI configuration files

`.json`
: for JSON configuration files

`.py`
: for Python configuration files

`.toml`
: for TOML configuration file

`.yml` or `.yaml`
: for YAML configuration files

When loading two or more configuration files the configurations will be merged:

- later values overwrite earlier values
- `dict`[9]-like objects are merged *recursively*

[9] https://docs.python.org/3/library/stdtypes.html#dict

- `list`[10] objects are by default completely replaced by later ones. When using `merge_lists="extend"` then later list extend earlier lists, when using `merge_lists="prepend"` then earlier lists extend later ones.

  This is done *non-recursively*.

2.7 Getting configuration variables

Get a – possibly interpolated – configuration variable's value with

```
value1 = config.getvar_s("key1")
value2 = config.getvar_s("key1.subkey2")
```

or equivalently with

```
value1 = config.getvarl_s("key1")
value2 = config.getvarl_s("key1", "subkey2")
```

Get a raw configuration variable's value with

```
value1_raw = config.getvar("key1")
value2_raw = config.getvarl("key1.subkey2")
```

or equivalently with

```
value1_raw = config.getvarl("key1")
value2_raw = config.getvarl("key1", "subkey2")
```

Because the configuration is not only a plain list of but a tree of key-value pairs you will want to fetch a nested configuration value using two access basic methods:

Configuration.getvar() and *Configuration.getvar_s()*

Use a single key variable where the invidual level keys are joined using a dot (`.`)

Configuration.getvarl() and *Configuration.getvarl_s()*

Use just positional Python arguments for each level key

Also there exist variants of the basic access methods that coerce returned variables into `int`[11] or `bool`[12] types (`Configuration.getintvar_s()`, `Configuration.getboolvar_s()`)

And with *Configuration.getfirstvar()*, *Configuration.getfirstvar_s()*, `Configuration.getfirstintvar_s()`, `Configuration.getfirstboolvar_s()` and `Configuration.getfirstfloatvar_s()` there exist variants that accept a *list* of possible variables names and return the first one that is found.

And again — with *Configuration.getfirstvarl()*, *Configuration.getfirstvarl_s()*, `Configuration.getfirstintvarl_s()`, `Configuration.getfirstboolvarl_s()` and `Configuration.getfirstfloatvarl_s()` there exist variants that accept a *list* of lists or tuples or dicts that describe possible variables names and return the first one that is found.

[10] https://docs.python.org/3/library/stdtypes.html#list
[11] https://docs.python.org/3/library/functions.html#int
[12] https://docs.python.org/3/library/functions.html#bool

For example – these methods for retrieving the first found variables can be used and are equivalent (Note that a caller that wants to use variables from a non-default namespace must use a sequence of dicts here):

```
value1 = config.getfirstvar_s("key1.subkey2", "key3.subkey4", default=None,␣
↪namespace=None)
value2 = config.getfirstvarl_s(*[["key1", "subkey2"], ["key3", "subkey4"]],␣
↪default=None)
value3 = config.getfirstvarl_s(*(("key1", "subkey2"), ("key3", "subkey4")),␣
↪default=None)
value4 = config.getfirstvarl_s(*[{"namespace": None, "path": ["key1", "subkey2"]},
↪{"namespace": None, "path": ("key3", "subkey4")}], default=None)
```

Looking at the example in chapter *YAML Files* – when calling `config.getvar_s("tree1.tree2.key4")` you will get the value `get this as 'tree1.tree2.key4'`.

Alternatively `config.getvarl_s("tree1", "tree2", "key4")` can be called with the very same result.

All four methods also perform direct *Variable Interpolation* and handle *Variable Namespaces* – yet in different ways. Filtering is not supported. So – the variable name arguments of *`Configuration.getvar()`* and *`Configuration.getvar_s()`* are of the form `[namespace:]variable` where for *`Configuration.getvarl()`* and *`Configuration.getvarl_s()`* the namespace is given as optional keyword parameter *namespace*.

Note: Special characters within namespace, key and filter names *must* be quoted (see *Quoting*) when using *`getvar()`* or *`getvar_s()`* to retrieve variables.

With *`getvarl()`* or *`getvarl_s()`* quoting is neither needed and not supported.

2.8 Direct Access to List Items

Direct access to list items is possible:

- Directly use the integer list index in *`getvarl()`* and its friends.
- Encode the index number to string format using the `~INDEX~` syntax and use *`getvar()`* and its friends.

  This syntax is also supported for variable interpolations.

Negative indexes are supported with Python semantics.

Examples:

- `config.getvarl_s("mylist", 0)` or `config.getvar_s("mylist.~0~)"`
- `config.getvarl_s("mylist", -1)` or `config.getvar_s("mylist.~-1~")`

This also works when using *Jailed Configurations*.

Use *Quoting* for the `~` character (`%x7e`) when the `~INDEX~` syntax should not be interpreted as list index but as key string.

2.9 Deletions

By using the special value `{{::DEL::}}` the corresponding key-value pair is deleted when merging is done.

2.10 Comments

By default all keys beginning with `__comment` or `__doc` are filtered out and not given to the application. This allows comments in JSON files – but is not restricted to JSON files only.

For all types of configuration files their respective standard comments are allowed too.

2.11 Variable Namespaces

Currently there are 6 namespaces:

1. The unnamed namespace (which is also default).

   All the configuration variables are part of this namespace.

   See also:

   Quoting

2. The namespace `ref` to be used for configuration references.

   This is a namespace that is handled special within "ConfigMix".

   Must be Filters are **not** supported.

   Think of them as symbolic links.

   See also chapter *Configuration tree references*.

3. The namespace `OS`

   Available functions:

   cwd
   : Contains the current working directory of the process

   node
   : Contains the current node's computername (or whatever `platform.node()`[13] returns)

4. The namespace `SYS`

   Available functions:

   executable
   : Contains the content of the current running Python's `sys.executable`[14].

   prefix
   : Contains the content of the current running Python's `sys.prefix`[15].

[13] https://docs.python.org/3/library/platform.html#platform.node
[14] https://docs.python.org/3/library/sys.html#sys.executable
[15] https://docs.python.org/3/library/sys.html#sys.prefix

base_prefix
Contains the content of the current running Python's `sys.base_prefix`[16].

Raises `KeyError`[17] if the attribute is not available.

platform
Contains the content of the current running Python's `sys.platform`[18].

5. The namespace `ENV`

   This namespace contains all the environment variables as they are available from `os.environ`[19].

6. The namespace `PY`

   Contains selected values from the running Python:

   version
   The return value of `platform.python_version()`[20]

   version_maj_min
   Just the major and minor version of the running Python (`.` separated)

   version_maj
   Just the major version of the running Python

   implementation
   The return value of `platform.python_implementation()`[21]

7. The namespace `AWS`

   Contains some metadata for AWS instances when running from within AWS:

   `metadata.instance-id`

   `metadata.placement.region`

   `metadata.placement.availability-zone`

   dynamic.instance-identity.region
   and all other properties of the instance-identity document (e.g. `instanceId`, `instanceType`, `imageId`, `pendingTime`, `architecture`, `availabilityZone`, `privateIp`, `version` et al.).

[16] https://docs.python.org/3/library/sys.html#sys.base_prefix
[17] https://docs.python.org/3/library/exceptions.html#KeyError
[18] https://docs.python.org/3/library/sys.html#sys.platform
[19] https://docs.python.org/3/library/os.html#os.environ
[20] https://docs.python.org/3/library/platform.html#platform.python_version
[21] https://docs.python.org/3/library/platform.html#platform.python_implementation

2.11.1 Examples

Both

```
config.getvar("OS:cwd")
```

or

```
config.getvarl("cwd", namespace="OS")
```

yield the current working directory – just as `os.getcwd()`[22] does.

2.12 Variable Interpolation

Configuration variable values that are read with *Configuration.getvar_s()* or *Configuration.getvarl_s()* are subject to variable interpolation. The general syntactic pattern for this is:

```
{{[namespace:]variable[|filter[|filter...]]}}
```

or:

```
{{[namespace:]variable[|filter[,filter...]]}}
```

I.e.: between double curly braces an optional *namespace* name followed by a colon `:`, the *variable* and then zero or more filters, the first one introduced by a pipe symbol `|` the following ones introduced by a comma `,` or a pipe symbol `|`. The comma `,` should be preferred.

Variables are expanded *lately* at runtime – exactly when calling *Configuration.getvar_s()*, *Configuration.getvarl_s()*, *Configuration.substitute_variables_in_obj()* or *Configuration.interpolate_variables()*

Note: Special characters within namespace, key and filter names *must* be quoted (see *Quoting*) when using variable interpolation syntax.

Note: Commata `,` and pipe symbols `|` are not allowed within filter names.

2.12.1 Filter functions

Interpolated values can be processed through a series of filter functions:

```
{{my.variable|filter1|filter2|filter3}}
```

or:

```
{{my.variable|filter1,filter2,filter3}}
```

[22] https://docs.python.org/3/library/os.html#os.getcwd

Available filter functions are:

`urlquote`

`urlquote_plus`

`saslprep`

`normpath`

`abspath`

`posixpath`

`lower`

`upper`

Also available are special filter functions `None` and `Empty`. They are useful in variable interpolation context because they suppress possible lookup errors (aka `KeyError`[23]) and instead return with `None`[24] or an empty string.

2.12.2 Nested Interpolation (Filtering Only)

Generally, nested interpolation does *not* work. Something like `{{{{variable}}}}` does not work.

Also something like `{{{{path-variable|Empty}}/subdir|normpath}}` does not work as expected: *path-variable* would not get interpolated and *Empty* not be applied.

But – as a special case – a simplified form of nested evaluation is implemented for filters only. Instead of using the start tag `{{` and end tag `}}` it uses the special start tag `{{|` and end tag `|}}`.

With the syntax:

```
{{|<expression with variables (including filters)>|filter1[,filter2...]|}}
```

The `<expression with variables (including filers)` is interpolated first, and the complete result is feeded into the filter chain, beginning with `filter1`.

Note: Chaining filters is allowed only with the comma `,` as separating symbol. Using the pipe symbol `|` is not supported.

With this, the non-working example above can be expressed as `{{|{{path-variable|Empty}}/subdir|normpath|}}` and yields the expected result – with *normpath* applied to the now interpolated expression.

[23] https://docs.python.org/3/library/exceptions.html#KeyError
[24] https://docs.python.org/3/library/constants.html#None

2.12.3 Examples

```
{{OS:cwd|posixpath}}
```

expands to the current working directory as POSIX path: on Windows all backslashes are replaced by forward slashes.

```
{{ENV:PATH}}
```

expands to the current search path from the process environment.

```
{{PY:version}}
```

expands to the current running Python version (e.g. `3.6.4`).

```
{{PY:implementation|upper}}
```

expands to something like `CPYTHON` when using the standard Python interpreter written in C.

2.13 Configuration tree references

Syntax is `{{ref:#my.other.key}}`.

- Think of it as a sort of a symbolic link to other parts of the configuration tree.
- They occupy the special namespace `ref`.
- Note that they can not be quoted currently in variable interpolation syntax.
- No special handling when merging is done – merging is agnostic of tree references.
- Keys within *Configuration.getvar_s()*, *Configuration.getvar()*, *Configuration.getvarl()* and *Configuration.getvarl_s()* are handled.
- In *Configuration.getvar()* a reference handled only when it is the directly referenced value
- Recursive expansion in *Configuration.getvar_s()* and *Configuration.getvarl_s()*: beware of recursive (direct or indirect) tree references.
- References do work as root paths of *Jailed Configurations*.

2.14 Quoting

When using *Configuration.getvar()* and *Configuration.getvar_s()* and when retrieving values in the default namespace the namespace separator `:` or the hierarchy separator `.` are characters with a special meaning. When using *variable interpolation* the filter separator `|` is also special. To use them in key names they must be quoted.

Quoting is done with a variant of the well-known percent-encoding in URIs (**RFC 3986**[25]).

A percent-encoded character consists of the percent character `%`, followed by one of the characters `x`, `u` or `U`, followed by the two, four or eight hexadecimal digits of the unicode codepoint value

[25] https://datatracker.ietf.org/doc/html/rfc3986.html

of the character that is to be quoted. `x` must be followed by two hex digits, `u` by four and `U` by eight.

Example:

> The character `.` with the Unicode (and ASCII) value 46 (hex 0x2e) can be encoded as `%x2e` or `%u002e` or `%U0000002e`.

Note: Filters neeed no quoting – and quoting within filters is *not* supported.

Note: Quoting the `ref` namespace name does not work currently when used in variable interpolation syntax.

2.15 Jailed Configurations

With *configmix.config.Configuration.jailed()* you get a *jailed* (or *restricted*) configuration from a "normal" configuration.

Restriction is two-fold:

- The access to configuration variables in *config* is restricted to the configuration sub-tree that is configured in *path*.
- Not all access-methods of `Configuration` are implemented yet.

This is somewhat analogous to a *chroot* environment for filesystems.

Note: The word "jail" is shamelessly stolen from FreeBSD jails.

Usage example:

```
import configmix

config = configmix.load("conf10.py")
assert not config.is_jail
value = config.getvar_s("tree1.tree2.key4")

jailed_config1 = config.jailed(rootpath="tree1.tree2")
assert jailed_config1.is_jail
assert jailed_config1.base is config
jvalue1 = jailed_config1.getvar_s("key4")

jailed_config2 = config.jailed(root=("tree1", "tree2"))
assert jailed_config2.is_jail
assert jailed_config2.base is config
jvalue2 = jailed_config.getvarl_s("key4")

assert value == jvalue1 == jvalue2 == "get this as 'tree1.tree2.key4'"
```

jvalue1 and *jvalue2* (and *value*) yield the very same value `get this as 'tree1.tree2.key4'` from the configuration.

All access methods in a jailed configuration automatically prepend the given *root path* in order to get the effective key into the base configuration.

It is possible to get a jailed configuration from an already jailed configuration. This sub-jail inherits the unjailed base configuration from the jailed configuration by default.

```
import configmix

config = configmix.load("conf10.py")
assert not config.is_jail
value = config.getvar_s("tree1.tree2.key4")

jailed_config1 = config.jailed(rootpath="tree1")
assert jailed_config1.is_jail
assert jailed_config2.base is config

jailed_config2 = jailed_config2.jailed(rootpath="tree2")
assert jailed_config2.is_jail
assert jailed_config2.base is config
jvalue2 = jailed_config.getvarl_s("key4")

assert value == jvalue2 == "get this as `tree1.tree2.key4'"
```

Note: A jailed configuration holds a strong reference to the unjailed base configuration.

Note: If a jail's root path points to a location with a variable substitutions the jail does not work: it is not possible to expand the substitution.

Using `ref` namespaces instead works: think of symbolic links.

2.16 Custom filename extensions and custom loaders

If you want to have custom configuration file extensions and/or custom loaders for custom configuration files you have various possibilities:

Associate an additional new extension (e.g. ".conf") with an existing configuration file style (e.g. YAML):

```
configmix.set_assoc("*.conf", configmix.get_assoc("*.yml"))
```

Allow only files with extension ".cfg" in INI-style – using the default loader for INI-files:

```
configmix.clear_assoc()
configmix.set_assoc("*.cfg", configmix.get_default_assoc("*.ini"))
```

Only a new configuration file style:

```
def my_custom_loader(filename):
    ...
```

(continues on next page)

(continued from previous page)

```
    return some_dict_alike

configmix.mode_loaders["myconfmode"] = my_custom_loader
configmix.clear_assoc()
configmix.set_assoc("*.my.configuration", "myconfmode")
```

If *clear_assoc()* will not be called then just a *new* configuration file style will be installed.

To select the loader not by extension but by an Emacs-compatible mode declaration (e.g. `mode: yaml`) in the first two lines of a file use:

```
configmix.set_assoc("*", configmix.try_determine_filemode)
```

CHAPTER 3

Package API Documentation

3.1 Package `configmix`

A library for helping with configuration files.

Author
Franz Glasner

License
BSD 3-Clause “New” or “Revised” License. See LICENSE.txt for details.

ID
@(#) $Header$

configmix.**load**(**files*, ***kwargs*)

Load the given configuration files, merge them in the given order and return the resulting configuration dictionary.

Parameters

- **files** – the filenames of the configuration files to read and merge; if a filename starts with `<dir>` then the name is interpreted as directory and all files are loaded in sorted order (non-resursively, ignoring unknown filetypes)
- **defaults** (*dict-alike or None*) – optional configuration dictionary with some default settings where the settings from *files* are merged into
- **extras** (*dict-alike or None*) – optional configuration dictionary that will applied last

  Use this for example to overwrite configuration file settings from commandline arguments.
- **strict** (*bool*[26]) – enable strict parsing mode for parsers that support it (e.g. to prevent duplicate keys)
- **merge_lists** (*str*[27] *or None*) – When `None` then lists will be overwritten by the merge process. When `extend` then lists will be extended instead. This parameter is passed to *merge()*.

Returns
the configuration

Return type
Configuration

configmix.**safe_load**(**files*, ***kwargs*)

Analogous to *load()* but do merging with *safe_merge()* instead of *merge()*

configmix.**try_determine_filemode**(*filename*)

Try to determine an explicitely given filemode from an Emacs-compatible mode declaration (e.g. `mode=python`).

Parameters
filename (*str*[28]) –

Returns
the found mode string or *None*

Return type
str[29] or None

Only the first two lines are searched for.

Conveniently to be used in calls to *set_assoc()* to determine the file-mode by content instead of filename extension.

[26] https://docs.python.org/3/library/functions.html#bool
[27] https://docs.python.org/3/library/stdtypes.html#str

configmix.**DEFAULT_MODE_LOADERS = {'-*- ignore -*-': <function _load_ignore>, '-*-ignore-*-': <function _load_ignore>, 'conf': <function _load_ini>, 'conf-toml': <function _load_toml>, 'conf-windows': <function _load_ini>, 'ini': <function _load_ini>, 'javascript': <function _load_json>, 'json': <function _load_json>, 'python': <function _load_py>, 'toml': <function _load_toml>, 'yaml': <function _load_yaml>}**

Default associations between file modes and loader functions

configmix.**DEFAULT_ASSOC = [('*.yml', 'yaml'), ('*.yaml', 'yaml'), ('*.json', 'json'), ('*.py', 'python'), ('*.ini', 'conf'), ('*.toml', 'toml')]**

The builtin default associations of filename extensions with file modes – in that order.

The "mode" part may be a string or a callable with a filename parameter that returns the mode string for the file or *None* if it can not determined.

configmix.**USE_DEFAULT_ASSOC = <object object>**

Marker for the default association for an extension.

To be used in *set_assoc()*.

configmix.**get_default_assoc**(*pattern*)

Return the default file-mode association for the fnmatch[30] pattern *pattern*.

Raises

KeyError[31] if the *pattern* is not found.

configmix.**mode_loaders = {'-*- ignore -*-': <function _load_ignore>, '-*-ignore-*-': <function _load_ignore>, 'conf': <function _load_ini>, 'conf-toml': <function _load_toml>, 'conf-windows': <function _load_ini>, 'ini': <function _load_ini>, 'javascript': <function _load_json>, 'json': <function _load_json>, 'python': <function _load_py>, 'toml': <function _load_toml>, 'yaml': <function _load_yaml>}**

All configured associations between file modes and loader functions.

See *DEFAULT_MODE_LOADERS*.

configmix.**clear_assoc**()

Remove all configured loader associations.

The *DEFAULT_ASSOC* are **not** changed.

configmix.**get_assoc**(*pattern*)

Return the default loader for the fnmatch[32] pattern *pattern*.

Raises

KeyError[33] if the *pattern* is not found.

configmix.**set_assoc**(*fnpattern*, *mode*, *append=False*)

Associate a fnmatch[34] style pattern *fnpattern* with a file-mode *mode* that determines what will be called when *load()* encounters a file argument that matches *fnpattern*.

[28] https://docs.python.org/3/library/stdtypes.html#str
[29] https://docs.python.org/3/library/stdtypes.html#str
[30] https://docs.python.org/3/library/fnmatch.html#module-fnmatch
[31] https://docs.python.org/3/library/exceptions.html#KeyError
[32] https://docs.python.org/3/library/fnmatch.html#module-fnmatch
[33] https://docs.python.org/3/library/exceptions.html#KeyError

Parameters

- **fnpattern** (*str*[35]) – the fnmatch[36] pattern to associate a loader with
- **mode** (*str*[37] *or callable*) – a mode string or a callable that accepts a *filename* argument and returns a file-mode for the given file (or *None*)
- **append** (*bool*[38]) – If *False* (which is the default) then this function inserts the given pattern at the head position of the currently defined associations, if *True* the pattern will be appended

The OS specific case-sensitivity behaviour of fnmatch.fnmatch()[39] applies (i.e. os.path.normpath()[40] will be called for both arguments).

If *loader* is *USE_DEFAULT_ASSOC* then the default association from *DEFAULT_ASSOC* will be used – if any.

configmix.**del_assoc**(*fnpattern*)

Remove all associations for *fnpattern*.

Parameters

fnpattern (*str*[41]) – the fnmatch[42] pattern to associate a loader with

configmix.**merge**(*user*, *default*, *filter_comments=True*, *merge_lists=None*)

Logically merge the configuration in *user* into *default*.

Parameters

- **user** (Configuration) – the new configuration that will be logically merged into *default*
- **default** (Configuration) – the base configuration where *user* is logically merged into
- **filter_comments** (*bool*[43]) – flag whether to filter comment keys that start with any of the items in *COMMENTS*
- **merge_lists** (*str*[44] *or None*) – When None then lists will be overwritten by the merge process. When extend then lists will be extended instead.

Returns

user with the necessary amendments from *default*. If *user* is None then *default* is returned.

Note: The configuration in *user* is augmented/changed **inplace**.

The configuration in *default* will be changed **inplace** when filtering out comments (which is the default).

---

[34] https://docs.python.org/3/library/fnmatch.html#module-fnmatch
[35] https://docs.python.org/3/library/stdtypes.html#str
[36] https://docs.python.org/3/library/fnmatch.html#module-fnmatch
[37] https://docs.python.org/3/library/stdtypes.html#str
[38] https://docs.python.org/3/library/functions.html#bool
[39] https://docs.python.org/3/library/fnmatch.html#fnmatch.fnmatch
[40] https://docs.python.org/3/library/os.path.html#os.path.normpath
[41] https://docs.python.org/3/library/stdtypes.html#str
[42] https://docs.python.org/3/library/fnmatch.html#module-fnmatch

If a value in *user* is equal to *constants.DEL_VALUE* ({{::DEL::}}) the corresponding key will be deleted from the merged output.

From http://stackoverflow.com/questions/823196/yaml-merge-in-python

configmix.**safe_merge**(*user*, *default*, *filter_comments=True*, *merge_lists=None*)

A more safe version of *merge()* that makes deep copies of the returned container objects.

Contrary to *merge()* no given argument is ever changed inplace. Every object from *default* is decoupled from the result – so changing the *default* configuration later does not propagate into a merged configuration later.

3.2 Module configmix.compat

Some minimal compatibility shim between Python2 and Python3

configmix.compat.**text_to_native_os_str**(*s*, *encoding=None*)

configmix.compat.**native_os_str_to_text**(*s*, *encoding=None*)

configmix.compat.**u**(*s*, *encoding='utf-8'*)

configmix.compat.**u2fs**(*s*, *force=False*)

Convert a text (Unicode) string to the filesystem encoding.

Note: The filesystem encoding on Python 3 is a Unicode text string. The function is a noop when called on Python 3.

Note: If *s* is already a byte string be permissive and return *s* unchanged.

---

configmix.compat.**uchr**(*n*)

configmix.compat.**n**(*s*, *encoding='utf-8'*)

configmix.compat.**str_and_u**(*v*)

Convert the value in *v* of any type to a native string and then to text (Unicode)

3.3 Module configmix.config

The unified configuration dictionary with attribute support or variable substitution.

class configmix.config.**Configuration**(**args*, ***kwds*)

The configuration dictionary with attribute support and variable interpolation/substitution.

Note: When retrieving by attribute names variables will be interpolated.

---

[43] https://docs.python.org/3/library/functions.html#bool
[44] https://docs.python.org/3/library/stdtypes.html#str

getvar()

For documentation and the signature see *py_getvar()*.

This method is an alias of *py_getvar()* or *fast_getvar()* – depending on the availability of the *configmix._speedups* module.

py_getvar(*varname*, *default=<object object>*)

Get a variable of the form `[ns:][[key1.]key2.]name` - including variables from other namespaces.

No variable interpolation is done and no filters are applied.

Special characters (e.g. `:` and `.`) must be quoted when using the default namespace.

See also *quote()*.

Pure-Python implementation.

fast_getvar()

Implemented in C in *configmix._speedups*.

getvar_s()

For documentation and the signature see *py_getvar_s()*.

This method is an alias of *py_getvar_s()* or *fast_getvar_s()* – depending on the availability of the *configmix._speedups* module.

py_getvar_s(*varname*, *default=<object object>*)

Get a variable - including variables from other namespaces.

varname is interpreted as in *getvar()*. But variables will be interpolated recursively within the variable values and filters are applied.

For more details see chapter *Variable Interpolation*.

Pure-Python implementation.

fast_getvar_s()

Implemented in C in *configmix._speedups*.

getvarl()

For documentation and the signature see *py_getvarl()*.

This method is an alias of *py_getvarl()* or *fast_getvarl()* – depending on the availability of the *configmix._speedups* module.

py_getvarl(**path*, ***kwds*)

Get a variable where the hierarchy is given in *path* as sequence and the namespace is given in the *namespace* keyword argument.

No variable interpolation is done and no filters are applied.

Quoting of *path* and *namespace* is *not* needed and wrong.

Pure-Python implementation.

fast_getvarl()

Implemented in C in *configmix._speedups*.

getvarl_s()

For documentation and the signature see *py_getvarl_s()*.

This method is an alias of *py_getvarl_s()* or *fast_getvarl_s()* – depending on the availability of the *configmix._speedups* module.

py_getvarl_s(**path*, ***kwds*)

Get a variable - including variables from other namespaces.

path and *namespace* are interpreted as in *getvarl()*. But variables will be interpolated recursively within the variable values and filters are applied.

For more details see chapter *Variable Interpolation*.

Pure-Python implementation.

fast_getvarl_s()

Implemented in C in *configmix._speedups*.

interpolate_variables()

For documentation and the signature see *py_interpolate_variables()*.

This method is an alias of *py_interpolate_variables()* or *fast_interpolate_variables()* – depending on the availability of the *configmix._speedups* module.

py_interpolate_variables(*s*)

Expand all variables in the single string *s*

Pure-Python implementation.

fast_interpolate_variables()

Implemented in C in *configmix._speedups*.

is_jail = False

Flag to show that this is not a jail for another configuration

clear_cache()

Clear the internal lookup cache and the interpolation cache

disable_cache()

Disable the internal lookup cache and the interpolation cache

enable_cache()

Enable the internal lookup cache and the interpolation cache.

The caches are empty after enabling.

__getitem__(*key*)

Mapping and list interface that forwards to *getvarl_s()*

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

Mapping interface that forwards to *getvarl_s()*

__contains__(*key*)

Containment test

`getitem_ns`(*key*)

Just forward to the *original* `dict.__getitem__()`.

No variable interpolation and key path access.

`items`()

Items without interpolation

`values`()

Values without interpolation

`getkeysl`(**path*, ***kwds*)

Yield the keys of a variable value.

Return type
A generator

Raises
`KeyError`[45] –

Note: Dictionary keys are not subject to interpolation.

`getfirstvarl`(**paths*, ***kwds*)

A variant of *getvarl()* that returns the first found variable in the *paths* list.

Every item in *paths* is either a *tuple* or *list* or a *dict*. If the path item is a *dict* then it must have two keys "namespace" and "path". If the path item is a *list* or *tuple* then the namespace is assumed to be *None*.

Note that a caller that wants to use variables from a non-default namespace must use a sequence of dicts.

No variable interpolation is done and no filters are applied.

Quoting of anything in *paths* is *not* needed and wrong.

`getkeys`(*varname*)

Yield all the keys of a variable value.

Return type
A generator

Raises
`KeyError`[46] –

Note: Dictionary keys are not subject to interpolation.

`getfirstvar`(**varnames*, ***kwds*)

A variant of *getvar()* that returns the first found variable in the list of given variables in *varnames*.

`getfirstvarl_s`(**paths*, ***kwds*)

A variant of *getfirstvarl()* that does variable interpolation.

paths and *kwds* are interpreted as in *getfirstvarl()*. But variables will be interpolated recursively within the variable values and filters are applied.

For more details see chapter *Variable Interpolation*.

getfirstvar_s(**varnames*, ***kwds*)

A variant of *getvar_s()* that returns the first found variable in the list of given variables in *varnames*.

expand_if_reference(*v*)

Check whether *v* is a configuration reference and – if true – then expand it.

v must match the pattern `{{ref:<REFERENCE>}}`

All non-matching texttypes and all non-texttypes are returned unchanged.

Raises
KeyError[47] – If the reference cannot found

expand_ref_uri(*uri*)

Raises
KeyError[48] – If the reference URI is not found

try_get_reference_uri(*v*)

Check whether *v* is a configuration reference and – if true – return the configuration path where the reference points to.

If *v* is not a text type or not a reference return *None*.

Does not check whether the referenced configuration object exists.

Return type
None or str[49]

substitute_variables_in_obj(*obj*)

Recursively expand variables in the object tree *obj*.

jailed(*rootpath=None*, *root=None*, *bind_root=True*)

Return a "jailed" configuration of the current configuration.

Parameters

- **rootpath** (*list*[50] *or* *tuple*[51]) – a sequence of strings (or objects) that shall emcompass the chroot-like jail of the returned configuration
- **root** (*str*[52]) – a string path expression that shall encompass the chroot-like jail of the returned configuration
- **bind_root** (*bool*[53]) – if you do a *rebind()* just after creation of a jailed config you can set *bind_root* to *False*; otherwise use the default

Returns
a jailed (aka restricted) configuration

Return type
_JailedConfiguration

Exactly one of *rootpath* or *root* must be given.

iter_jailed(*rootpath=None*, *root=None*)

Iterator that yields properly jailed configurations.

rootpath or *root* must refer to a *list* or *dict* container.

extract_new_config(**path*, ***kwds*)

Get the value at *path* and make a new *Configuration* from it.

The new configuration is a deepcopy and completely independent of the source configuration.

copy_new_config_without(**path*)

Copy the current configuration but leave out the value at key *path*.

The new configuration is a deepcopy and completely independent of the source configuration.

Note: Currently only a "simple" *path* with length 1 is supported. References are not supported.

---

class configmix.config.**CoercingMethodsMixin**

Mixin to provide some common implementations for retrieval methods that convert return values to a fixed type (int, bool, float).

Both *Configuration* and *_JailedConfiguration* use this mixin.

getintvarl_s(**path*, ***kwds*)

Get a (possibly substituted) variable and coerce text to a number.

getfirstintvarl_s(**paths*, ***kwds*)

Get a (possibly substituted) variable and coerce text to a number.

getintvar_s(*varname*, *default=<object object>*)

Get a (possibly substituted) variable and coerce text to a number.

getfirstintvar_s(**varnames*, ***kwds*)

A variant of *getintvar_s()* that returns the first found variable in the list of given variables in *varnames*.

getboolvarl_s(**path*, ***kwds*)

Get a (possibly substituted) variable and convert text to a boolean

getfirstboolvarl_s(**paths*, ***kwds*)

Get a (possibly substituted) variable and convert text to a boolean

[45] https://docs.python.org/3/library/exceptions.html#KeyError
[46] https://docs.python.org/3/library/exceptions.html#KeyError
[47] https://docs.python.org/3/library/exceptions.html#KeyError
[48] https://docs.python.org/3/library/exceptions.html#KeyError
[49] https://docs.python.org/3/library/stdtypes.html#str
[50] https://docs.python.org/3/library/stdtypes.html#list
[51] https://docs.python.org/3/library/stdtypes.html#tuple
[52] https://docs.python.org/3/library/stdtypes.html#str
[53] https://docs.python.org/3/library/functions.html#bool

getboolvar_s(*varname*, *default=<object object>*)

Get a (possibly substituted) variable and convert text to a boolean

getfirstboolvar_s(**varnames*, ***kwds*)

A variant of *getboolvar_s()* that returns the first found variable in the list of given variables in *varnames*.

getfloatvarl_s(**path*, ***kwds*)

Get a (possibly substituted) variable and convert text to a float

getfirstfloatvarl_s(**path*, ***kwds*)

Get a (possibly substituted) variable and convert text to a float

getfloatvar_s(*varname*, *default=<object object>*)

Get a (possibly substituted) variable and convert text to a float

getfirstfloatvar_s(*varname*, *default=<object object>*)

Get a (possibly substituted) variable and convert text to a float

While not instantiable directly, but only by *Configuration.jailed()*, the API use will want to know its interface:

class configmix.config.**_JailedConfiguration**(**path*)

A jailed and restricted variant of *Configuration*.

Restriction is two-fold:

- The access to configuration variables in *config* is restricted to the configuration sub-tree that is configured in *path*.
- Not all access-methods of *Configuration* are implemented yet.

See also:

Jailed Configurations

Note: There is no namespace support.

Note: Do not call the constructor directly. Instantiate a jailed configuration from the parent configuration's *jailed()* factory method.

---

is_jail = True

Flag to show that this is a jail for another configuration

property base

Ask for the base (aka parent) configuration".

This configuration is always unjailed.

rebind(*new_base*)

Bind the jail to a new unjailed configuration *new_base*.

The new configuration base also must have an existing path to the root.

Parameters
new_base (Configuration) – the new base

__getattr__(*name*)
Attribute-style access.

Result values are interpolated (i.e. forwarded to *getvarl_s()*)

__getitem__(*key*)
Mapping and list interface that forwards to *getvarl_s()*

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

__contains__(*key*)
Containment support for containers

getvarl(**path*, ***kwds*)

getkeysl(**path*, ***kwds*)

getfirstvarl(**paths*, ***kwds*)

getvarl_s(**path*, ***kwds*)

getfirstvarl_s(**paths*, ***kwds*)

getvar(*varname*, ***kwds*)

getkeys(*varname*)

getfirstvar(**varnames*, ***kwds*)

getvar_s(*varname*, ***kwds*)

getfirstvar_s(**varnames*, ***kwds*)

__iter__()
Iteration support for containers

__len__()
Length support for containers

iter_jailed()
Iteration support for containers which yields properly jailed sub-jails.

Only supported for type *list* or type *dict* jails.

__bool__()
Map- and list-style evaluation in boolean context

jailed(*rootpath=None*, *root=None*, *bind_root=True*)
Return a "jailed" configuration that effectively is a subjail of the current jail

For a more complete description see *Configuration.jailed()*.

Some public helper functions:

configmix.config.**quote**()

For documentation and the signature see *py_quote()*.

This function is an alias of *py_quote()* or *fast_quote()* – depending on the availability of the *configmix._speedups* module.

configmix.config.**fast_quote**()

Implemented in C in *configmix._speedups*.

configmix.config.**py_quote**(*s*)

Replace important special characters in string *s* by replacing them with `%xNN` where *NN* are the two hexadecimal digits of the character's unicode codepoint value.

Handled are the important special chars: `%`, `.`, `:`, `#`; `'`, `"`, `|`, `{`, `}`, `[` and `]`.

See also the *Quoting* section.

Pure-Python implementation.

configmix.config.**unquote**()

For documentation and the signature see *py_unquote()*.

This function is an alias of *py_unquote()* or *fast_unquote()* – depending on the availability of the *configmix._speedups* module.

configmix.config.**fast_unquote**()

Implemented in C in *configmix._speedups*.

configmix.config.**py_unquote**(*s*)

Unquote the content of *s*: handle all patterns `%xNN`, `%uNNNN` or `%UNNNNNNNN`.

This is the inverse of *quote()*.

Pure-Python implementation.

configmix.config.**pathstr2path**()

For documentation and the signature see *py_pathstr2path()*.

This function is an alias of *py_pathstr2path()* or *fast_pathstr2path()* – depending on the availability of the *configmix._speedups* module.

configmix.config.**fast_pathstr2path**()

Implemented in C in *configmix._speedups*.

configmix.config.**py_pathstr2path**(*varname*)

Parse a dot-separated path string *varname* into a tuple of unquoted path items

Parameters
varname (*str*[54]) – The quoted and dot-separated path string

Returns
The unquoted and parsed path items

Return type
tuple[55]

Used e.g. by *getvar()*, *getvar_s()* and *jailed()*.

The returned value is suitable as input for *getvarl()*, *getvarl_s()* and friends.

An empty *varname* returns an empty tuple.

Pure-Python implementation.

3.4 Module configmix.constants

Some important public contants

configmix.constants.**COMMENTS = ['__comment', '__doc']**

Prefixes for comment configuration keys that are to be handled as comments

configmix.constants.**DIR_PREFIX = '<dir>'**

Prefix for configuration values to read other configuration files from given directory

configmix.constants.**DEL_VALUE = '{{::DEL::}}'**

Value for configuration items to signal that the corresponding key-value is to be deleted when configurations are merged.

Despite having "interpolation" syntax this value will **never** be substituted.

configmix.constants.**REF_NAMESPACE = 'ref'**

Special internal namespace used for implementation of tree *references*

configmix.constants.**NONE_FILTER = 'None'**

The public name of the special *None* filter

configmix.constants.**EMPTY_FILTER = 'Empty'**

The public name of the special *None* filter

3.5 Module configmix.ini

Read INI-style configuration files.

class configmix.ini.**INIConfigParser**(*filename*, *executable=None*, *encoding=None*)

A case sensitive config parser that returns all-unicode string values.

read(*filenames*)

Not implemented. Use *read_file()* instead.

readfp(*fp*, *filename*)

Compatibility for older Python versions.

Use *read_file()* instead.

read_file(*fp*, *filename*)

Read from a file-like object *fp*.

The *fp* argument must be iterable (Python 3.2+) or have a *readline()* method (Python 2, <3.2).

[54] https://docs.python.org/3/library/stdtypes.html#str
[55] https://docs.python.org/3/library/stdtypes.html#tuple

getx(*section*, *option*)

Extended *get()* with some automatic type conversion support.

Default: Fetch as string (like `get()`).

If annotated with `:bool:` fetch as bool, if annotated with `:int:` fetch as int, if annotated with `:float:` fetch as float.

itemsx(*section*, *options*)

Get all the options given in *options* of section *section*. Fetch them with *getx()* in the order given.

Return a list of `(name, value)` pairs for each option in *options* in the given *section*.

items_as_dictx(*section*, *options*)

Similar to *itemsx()* but return a (possibly ordered) dict instead of a list of key-value pairs.

configmix.ini.**load**(*filename*, *extract=['config']*, *encoding='utf-8'*)

Load a single INI file and read/interpolate the sections given in *extract*.

Flattens the given sections into the resulting dictionary.

Then build a tree out of sections which start with any of the *extract* content value and a dot `.`.

The encoding of the file is given in *encoding*.

3.6 Module configmix.json

Read JSON-style configuration files.

configmix.json.**load**(*filename*, *encoding='utf-8'*)

Load a single JSON file with name *filename* and encoding *encoding*.

3.7 Module configmix.py

Read configuration settings from Python files.

configmix.py.**load**(*filename*, *extract=None*)

Load Python-style configuration files.

Files are loaded and **executed** with `exec()`[56] using an empty dict as global and local context.#

Parameters

- **filename** – the path to the configuration file
- **extract** – an optional list of variable names (keys) to extract into the resulting configuration dictionary

Returns

the configuration as dictionary

Return type
collections.OrderedDict[57] or ordereddict.OrderedDict or dict[58]

3.8 Module configmix.toml

Read TOML style configuration files.

configmix.toml.**load**(*filename*, *encoding='utf-8'*)
Load a single TOML file with name *filename* and encoding *encoding*.

Note: The TOML standard requires that all TOML files are UTF-8 encoded.

3.9 Module configmix.variables

Variable interpolation: implementation of namespaces and filters

configmix.variables.**add_varns**(*name*, *fn*)
Register a new variable namespace *name* and it's implementing function *fn*

..note:: This function checks that *name* is not the special
namespace *REF_NAMESPACE*.

configmix.variables.**lookup_varns**(*name*)
Lookup the variable namespace *name* and return it's implementing function

Parameters
name (*str*[59]) – the namespace name

Returns
the implementing function

Raises
KeyError[60] – if the namespace *name* doesn't exist

..note:: This function checks that *name* is not the special
namespace *REF_NAMESPACE*.

configmix.variables.**add_filter**(*name*, *fn*)
Register a variable filter function with name *name* and implementation *fn*.

name may *not* contain `,` and `|` characters.

Raises
ValueError[61] – If an invalid *name* is given

[56] https://docs.python.org/3/library/functions.html#exec
[57] https://docs.python.org/3/library/collections.html#collections.OrderedDict
[58] https://docs.python.org/3/library/stdtypes.html#dict
[59] https://docs.python.org/3/library/stdtypes.html#str
[60] https://docs.python.org/3/library/exceptions.html#KeyError
[61] https://docs.python.org/3/library/exceptions.html#ValueError

configmix.variables.**lookup_filter**(*name*)

Lookup a variable filter with name *name* and return it's implementation function

Parameters
name (*str*[62]) – the logical filter name

Returns
the implementing filter function

Raises
KeyError[63] – if the filter cannot be found

configmix.variables.**filter**(*name*)

Decorator for a filter function.

Example usage:

```
@filter("myfilter")
def myfilter_impl(appconfig, variable_value):
    filtered_value = ...
    return filtered_value
```

configmix.variables.**urlquote**(*config*, *v*)

Filter function to replace all special characters in string *v* using the %xx escape

configmix.variables.**urlquote_plus**(*config*, *v*)

Filter function to replace all special characters (including spaces) in string *v* using the %xx escape

configmix.variables.**saslprep**(*config*, *v*)

Filter function to perform a *SASLprep* according to **RFC 4013**[64] on *v*.

This is a Stringprep Profile for usernames and passwords

configmix.variables.**normpath_impl**(*config*, *v*)

Implementation of the *normpath* filter function

configmix.variables.**abspath_impl**(*config*, *v*)

Implementation of the *abspath* filter function

configmix.variables.**posixpath_impl**(*config*, *v*)

Implementation of the *posixpath* filter function

configmix.variables.**lower_impl**(*config*, *v*)

Implementation of the *lower* filter function

configmix.variables.**upper_impl**(*config*, *v*)

Implementation of the *upper* filter function

configmix.variables.**None_filter_impl**(*config*, *v*)

Identity.

The *None* filter is just a marker to not throw KeyError[65] but return *None*. It is a no-op within the filter-chain itself.

[62] https://docs.python.org/3/library/stdtypes.html#str
[63] https://docs.python.org/3/library/exceptions.html#KeyError
[64] https://datatracker.ietf.org/doc/html/rfc4013.html

configmix.variables.**Empty_filter_impl**(*config*, *v*)

Identity.

The *Empty* filter is just a marker to not throw KeyError[66] but return the empty string. It is a no-op within the filter-chain itself.

3.10 Module configmix.yaml

Simple wrapper for yaml to support all-unicode strings when loading configuration files.

class configmix.yaml.**ConfigLoader**(**args*, ***kwds*)

A YAML loader, which makes all !!str strings to Unicode. Standard PyYAML does this only in the non-ASCII case.

If an *OrderedDict* implementation is available then all "map" and "omap" nodes are constructed as *OrderedDict*. This is against YAML specs but within configuration files it seems more natural.

Initialize the scanner.

class configmix.yaml.**ConfigSafeLoader**(**args*, ***kwds*)

A safe YAML loader, which makes all !!str strings to Unicode. Standard PyYAML does this only in the non-ASCII case.

If an *OrderedDict* implementation is available then all "map" and "omap" nodes are constructed as *OrderedDict*. This is against YAML specs but within configuration files it seems more natural.

Initialize the scanner.

configmix.yaml.**load**(*stream*, *Loader=None*, *strict=False*)

Parse the given *stream* and return a Python object constructed from for the first document in the stream.

If *strict* is *True* then duplicate mapping keys within a YAML document are detected and prevented. If a *Loader* is given then *strict* does not apply.

configmix.yaml.**load_all**(*stream*, *Loader=None*, *strict=False*)

Parse the given *stream* and return a sequence of Python objects corresponding to the documents in the *stream*.

If *strict* is *True* then duplicate mapping keys within a YAML document are detected and prevented. If a *Loader* is given then *strict* does not apply.

configmix.yaml.**safe_load**(*stream*, *strict=False*)

Parse the given *stream* and return a Python object constructed from for the first document in the stream.

Recognizes only standard YAML tags and cannot construct an arbitrary Python object.

If *strict* is *True* then duplicate mapping keys within a YAML document are detected and prevented.

[65] https://docs.python.org/3/library/exceptions.html#KeyError
[66] https://docs.python.org/3/library/exceptions.html#KeyError

configmix.yaml.**safe_load_all**(*stream*, *strict=False*)

Return the list of all decoded YAML documents in the file *stream*.

Recognizes only standard YAML tags and cannot construct an arbitrary Python object.

If *strict* is *True* then duplicate mapping keys within a YAML document are detected and prevented.

3.11 Module configmix._speedups

This module is implemented in C using Python's C-API. It contains alternate implementations for some heavily used functions and/or methods. The module functions are not supposed to be called directly. Their signatures may or may not match their pure-Python equivalents because they may be called by appropriate tiny Python wrappers.

This module is optional.

The module is only available for CPython >= 3.7 and uses its stable API.

CHAPTER 4

Changes

All major changes over the versions are listed here. For breaking changes have a look at *Breaking Changes*, they are listed there in detail.

4.1 Changelog

4.1.1 Pre-1.0 Series

unreleased

- **[doc]** Build the PDF documentation with `lualatex` now.
- **[doc]** Some documentation tweaks.
- **[bugfix,doc]** Make the `intersphinx_mapping` configuration variable compatible with Sphinx 8.

  [misc] Begin a (BSD-)Makefile with some build helpers

0.23.3 (2023-12-07)

- **[feature]** Implement new method *`copy_new_config_without()`* for simple cases.
- **[bugfix,tests]** Add missing test data file(s).

0.23.2 (2023-12-04)

- **[feature]** Implement new method *extract_new_config()*.
- **[test]** Test fixes for the tests on Windows

0.23.1 (2023-10-31)

- **[bugfix,misc]** Implement real proper conformance to **PEP 491**[67]: Even is `Root-Is-Pure-lib: false` the wheel package data subdir `platlib` dictated installation into `platlib` previously.

0.23 (2023-10-30)

- **[feature]** Allow new merge strategies for lists: extend or prepend instead of replace. This is implemented with a new optional parameter `merge_lists` for *configmix.load()* or *configmix.merge()*.
- **[bugfix]** Allow again the installation of the pure-Python version for Python < 3.7.
- **[test]** Test fixes and enhancements

0.22 (2023-08-17)

- **[feature]** Nested interpolation for filters.
- **[feature]** Chaining filters can be done (and is indeed now the preferred way) with commas (`,`) in addition to the pipe symbol (`|`).

0.21.4 (2023-06-14)

- **[feature]** Implement a new `SYS` variable namespace with `executable`, `prefix`, `base_prefix` and `platform` as current content.

0.21.3 (2023-06-12)

- **[bugfix]** Fixed some format string errors in the YAML loader. These errors were only encountered in exception handlers.
- **[misc]** The installer got now an extra "sasl" that requires `passlib`. This package is needed by the filter `saslprep()`.
- **[misc]** Enhance compatibility with newer Python `setuptools` or `pip`: Create a basic `pyproject.toml`.
- **[doc]** Document that jails now support references at their roots: they act like a symbolic link.

[67] https://peps.python.org/pep-0491/

0.21.2 (2023-04-12)

- **[misc]** Test with PyYAML 6.0.

0.21.1 (2022-06-03)

- **[feature]** Enable indexed access to lists in the configuration using an access path string representation like `~NNN~`
- **[feature]** Implement methods *configmix.config.Configuration.iter_jailed()* and *configmix.config._JailedConfiguration.iter_jailed()* that yield properly jailed configurations for container items (for lists and dicts, no sets).
- **[feature]** Allow to enable and disable the internal caching
- **[feature]** Add support for using `tomllib`[68] (in Python's stdlib since 3.11) and `tomli` TOML packages. They are preferred if they are found to be installed.

  But note that the declared dependency for the `toml` extra nevertheless is the `toml` package. Because it is available for all supported Python versions. So use Python 3.11+ or install `tomli` manually if you want to use the alternate packages.
- **[bugfix]** For better consistency: use *.getvarl_s()* instead of *.getvarl()* in the implementation of *__len__()* in jailed configurations.
- **[bugfix]** For better TOML compatibility open TOML files with `encoding=""`.

0.21 (2022-06-03)

YANKED because of release errors.

0.20.5 (2022-03-07)

- **[bugfix]** The configuration value `{{::DEL::}}` is not subject to interpolation any more. This fixes the handling of these deletion markers when merging configurations: sometimes they were tried to be interpolated – and this failed.
- **[bugfix]** The merge logic should never interpolate variables. But some parts of the merge logic did this unintentionally.

0.20.4 (2022-01-17)

- **[bugfix]** In the C extension: make sure that a default is returned as-is and not as copy.
- **[misc]** Bring the exception messages from the C extension more in-line with the Python implementation.

[68] https://docs.python.org/3/library/tomllib.html#module-tomllib

0.20.3 (2022-01-12)

- **[feature]** Add some more functions to the optional C-extension module: *configmix.config.Configuration.getvarl_s()* and *configmix.config.Configuration.getvar()*.
- **[feature]** Some internal enhancements within the C-extension module.

0.20.2 (2022-01-11)

- **[bugfix]** The source distribution archive file did not contain any of the *__init__.py* files because of a bogus entry in `MANIFEST.in`.
- **[feature]** More speedsups by implementing *getvarl()* and *getvar_s()* within the C-extension also.

0.20.1 (2022-01-10)

- **[misc]** Add an optional C-extension with some speedups to often used functions and methods. Also allow to cross-build this modules for Windows with LLVM-11 and Ninja on POSIX systems.
- **[misc]** Some internal code refactoring. This also yields more consistency in interpreting the *varname* string arguments.

0.20 (2021-12-21)

- **[breaking]** Removed some unused keyword arguments from methods – also public ones:
  - *expand_ref_uri()*
  - *expand_if_reference()*
  - `_lookupref()`
- **[breaking]** Change methodname from *expand_variable()* to *configmix.config.Configuration.interpolate_variables()*
- **[misc]** Improved overall performance by a factor of 0.25 to 0.3 using algorithmic changes.
- **[misc]** Improved performance by internal caching.
- **[bugfix]** Implement *values()* and *items()* that yield non-interpolated configuration values. This is needed now because attribute access now yields interpolated results.

0.19.2 (2021-12-16)

- **[feature]** Implement "__len__()"-support for jailed configurations. Their container-like interface is now fairly complete.

0.19.1 (2021-12-15)

- **[feature]** Jailed configuration support proper evaluation in boolean context
- **[feature]** Proper iteration support for jailed configurations
- **[bugfix]** Fixes for proper exception formatting in many cases
- **[bugfix]** Proper "yield" support for older Python versions
- **[test]** Test fixes and enhancements

0.19.1b1 (2021-12-14)

- **[feature]** Attribute-style access also for jailed configurations
- **[bugfix}** Make exception formatting robust when there is a single parameter for "%"-style formatting and the single parameter happend to be a tuple; now it is wrapped into a tuple.

0.19 (2021-12-10)

- **[breaking] [feature}** Access to a configuration key using dict-level access now does variable interpolation
- **[feature]** Simple dict-style access for jailed configuration
- **[feature]** Optimized "__contains__()" implementation for jailed and unjailed configurations

0.18.1 (2021-12-10)

- **[feature]** Proper "repr()" for jailed configurations

0.18.1b1 (2021-12-09)

- **[bugfix]** Jailed configurations assumed that their "default" marker object is identical to the "default" marker object in the unjailed base configuration. This is not always true, especially if `rebind()` is used.

0.18 (2021-12-02)

- **[feature]** Allow empty variable names in some cases to get the root object of a configuration.
- **[feature]** Allow to get sub-jails from an already jailed configuration.
- **[feature]** Implement *getkeysl()* and *getkeys()* that return generators over all keys of a configuration value.

0.17 (2021-11-22)

- **[feature]** Complete the set of configuration retrieval methods for the jailed configuration.

0.17b2 (2021-11-19)

- **[feature]** All configuration objects carry a flag *is_jail* that allows to determine whether a configuration is jailed
- **[feature]** Allow a jailed configuration to be rebound to another unjailed configuration
- **[feature}** A public accessor property to the base configuration of a jailed configuration

0.17b1 (2021-11-19)

- **[feature]** Jailed (aka "restricted" or "rooted") configurations with *jailed()*

0.16.1 (2021-11-10)

- **[feature]** New access methods *getfirstvarl()*, *getfirstvarl_s()*, `getfirstintvarl_s()` `getfirstboolvarl_s()`, `getfirstfloatvarl_s()`
- **[feature]** New access method `getfirstfloatvar_s()`

0.16 (2021-07-11)

- **[feature]** New access methods *getfirstvar()*, *getfirstvar_s()*, `getfirstintvar_s()` and `getfirstboolvar_s()`

0.15.1 (2021-07-09)

- **[bugfix]** Handle the *default* keyword parameter in *configmix.config.Configuration.getvar()* properly.

0.15 (2021-06-25)

- **[feature]** New filter function `urlquote_plus()`
- **[feature]** New filter functions `None()` and `Empty()`. They are useful in variable interpolation context where they suppress possible lookup errors (aka `KeyError`[69]) and instead return with `None`[70] or an empty string.

0.14 (2021-05-10)

- **[breaking] [feature]** Allowed quoting of variable and namespace names.

  This is mostly important for variable names that contain `.`, `:` or `|` but probably useful for characters like "`` ` `` and # also.
- **[breaking] [misc]** Moved some important public constants from *configmix* into the *configmix.constants* module.
- **[feature]** Configuration tree references are implemented in the `ref` namespace
- **[feature]** Implemented new access methods *configmix.config.Configuration.getvarl()* and *configmix.config.Configuration.getvarl_s()*

0.13 (2021-04-21)

- **[feature]** All YAML load functions got a new optional keyword *strict* to detect and prevent duplicate keys within a single YAML document.

  The top-level load function also understands this flag and provides it to low-level-loaders that understand it.

0.12 (2020-12-07)

- **[feature]** Provide an `AWS` namespace to retrieve some AWS instance metadata.

0.11 (2020-10-05)

- **[feature]** Allow the deletion of key-value pairs while merging configurations.

  This is done by recognizing and handling the special configuration value `{{::DEL::}}`.

[69] https://docs.python.org/3/library/exceptions.html#KeyError
[70] https://docs.python.org/3/library/constants.html#None

0.10 (2020-09-10)

- **[feature]** Allow loading configuration files from directories when using the "<dir>" prefix in filenames.

  Unknown filetypes within these directories are ignored automatically.

- **[feature]** Implemented a function to delete an association: *configmix.del_assoc()*.

0.9 (2020-07-28)

- **[breaking] [feature]** Do not set "root", "self" and "here" variables any more. The old behaviour hindered proper automatic configuration on some PyPy configurations when using *genapplib*.

  Only the INI-parser did set this variables automatically.

0.8.1 (2020-07-08)

- **[bugfix]** Allow non-string keys when merging configurations.

0.8 (2020-07-08)

- **[breaking] [feature]** Do not implicitely convert a configuration value to text if the value is the result of just a variable expansion.

0.7.4 (2020-05-21)

- **[feature]** Implemented new namespace function `OS:node` to return the node's computer-name.
- **[bugfix]** The `OS` namespace lookup did not handle non-existing variables properly and ignored the *default* parameter.

0.7.3 (2020-05-13)

- No code changes.

0.7.2 (2019-05-13)

- **[feature]** Implemented a loader with key `-*-ignore-*-` effectively ignores the contents of given file. No file extensions are by default associated with this loader.

0.7.1 (2019-05-10)

- **[feature]** *configmix.load()* and *configmix.safe_load()* got a new keyword argument *extras* that (if given) will be used as the *last* configuration dictionary to be merged into the configuration.

  This can be used to overwrite configuration file settings from commandline arguments.

- **[bugfix]** *configmix.safe_load()* did some preliminary unsafe merges from *defaults* and an extra additional unneeded merge.

0.7 (2019-05-06)

- **[breaking]** Additional or alternative loaders can be installed by changing the *configmix.mode_loaders* dictionary directly.
- **[breaking]** The public functions to associate filename extensions to filemodes have been renamed to *configmix.set_assoc()*, *configmix.get_assoc()*, *configmix.clear_assoc()*, *configmix.get_default_assoc()*.

  The filemodes must be keys in the *configmix.mode_loaders* dictionary.

- **[breaking] [feature]** The associations from filename extensions to parsers are `fnmatch`[71] style patterns now.

  Calling *configmix.set_assoc()* by default prepends to the currently defined associations and therefore gets the highest priority. Appending is possible also.

- **[feature]** *configmix.load()* and *configmix.safe_load()* got a keyword argument *defaults* that allow the provision of an already existing default configuration into which all additional configuration settings are merged into.
- **[feature]** Added support for TOML style configuration files. This needs the external package `toml` (from https://github.com/uiri/toml).

0.6 (2019-03-14)

- **[breaking] [feature]** Reimplemented *configmix.safe_merge()* to do a deepcopy of all source configurations when merging. Previously is was sort of a shallow copy.
- **[breaking] [feature]** The default file encoding when reading INI style files with *configmix.ini.load()* is now "UTF-8". Previously it was undefined and therefore dependent on the user's locale.

  An *encoding* keyword argument can be specified explicitely now.

- **[breaking] [feature]** Support comment-like key-value pairs with configuration keys starting with `__doc` or `__comment`.
- **[misc]** Use the filesystem encoding where appripriate.
- **[doc]** Begin the documentation with Sphinx[72]
- **[test]** Begin formal unittests

[71] https://docs.python.org/3/library/fnmatch.html#module-fnmatch
[72] http://www.sphinx-doc.org

- **[feature]** Build a tree of configuration settings from INI files
- **[feature]** Support JSON formatted files as configuration files also (suffix ".json").
- **[feature]** Allow custom configuration filename extensions and custom loaders that can handle custom configuration file syntax styles.

0.5 (2016-04-19)

- **[feature]]** First really used release.

4.2 Breaking Changes

0.20

- Removed some unneeded keyword arguments from methods – also public ones:
  - Configuration.expand_ref_uri()
  - Configuration.expand_if_reference()
  - Configuration._lookupref()
- Change methodname from *expand_variable()* to *configmix.config.Configuration.interpolate_variables()*

0.19

- Dict-level access to a configuraiton now does variable interpolation

0.14

- Allow quoting of variable and namespace names
- Move some important public constants from *configmix* into the *configmix.constants* module.

These are technically a breaking changes while the author does not believe that any of the current clients is affected by both changes.

0.9

- Do not set "root", "self" and "here" variables any more. This hinders proper automatic configuration on some PyPy configurations when using *genapplib*.

  While technically a breaking change no known client is known to rely on the previous behaviour.

  Any only the INI-parser did set this variables automatically.

0.8

- Do not implicitely convert a configuration value to text if the value is the result of just a variable expansion.

  While technically a breaking change no known client is known to rely on the previous behaviour.

0.7

- A major overhaul of how filename extensions are associated with loaders has been done:
  - Filename extensions in `fnmatch`[73] style are associated with file-mode strings.

    These file-mode strings are associated with loader functions separately via the mapping *`configmix.mode_loaders`*.
  - *`configmix.set_assoc()`*, *`configmix.get_assoc()`*, *`configmix.clear_assoc()`* and *`configmix.get_default_assoc()`* are the new names for the old *set_loader()*, *get_loader()*, *clear_loader()* and *get_default_loader()* functions. They are used for associating `fnmatch`[74] style filename patterns to file-mode strings. Previously they associated loader functions directly.
  - *`configmix.set_assoc()`* now requires a `fnmatch`[75] style pattern instead of just a file extension string (i.e. a plain trailer). The previons dictionary with mapping from filename extensions to loader callables is now a list of tuples containing the `fnmatch`[76] style pattern and the corresponding loader callable.

0.6

- *`configmix.safe_merge()`* does now a deepcopy of all source configurations when merging. Changes in configuration instances afterwards will not be reflected in the merged configuration any more.

  The public signature of *`configmix.safe_merge()`* has *not* changed.
- The default file encoding when reading INI style files with *`configmix.ini.load()`* is now "UTF-8". Previously it was undefined and therefore dependent on the user's locale.

[73] https://docs.python.org/3/library/fnmatch.html#module-fnmatch
[74] https://docs.python.org/3/library/fnmatch.html#module-fnmatch
[75] https://docs.python.org/3/library/fnmatch.html#module-fnmatch
[76] https://docs.python.org/3/library/fnmatch.html#module-fnmatch

CHAPTER 5

Authors

The package *configmix* is written and maintained by Franz Glasner.

CHAPTER 6

License

The license is the 3-clause BSD license.

```
Copyright (c) 2015-2023, Franz Glasner
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the
   distribution.

3. Neither the name of the copyright holder nor the names of its
   contributors may be used to endorse or promote products derived
   from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# BSD 3-Clause "New" or "Revised" License
```

CHAPTER 7

TODO List

Python Module Index

C

Index

Symbols

A

B

C

D

Q

R

S

T

U

V