

---

# **configconfig**

*Release 0.6.2*

**Load and validate YAML configuration files.**

**Dominic Davis-Foster**

**Apr 25, 2024**



## Contents

|          |                                |           |
|----------|--------------------------------|-----------|
| <b>1</b> | <b>Installation</b>            | <b>1</b>  |
| 1.1      | from PyPI . . . . .            | 1         |
| 1.2      | from Anaconda . . . . .        | 1         |
| 1.3      | from GitHub . . . . .          | 1         |
| <b>2</b> | <b>configconfig.autoconfig</b> | <b>3</b>  |
| 2.1      | Usage . . . . .                | 3         |
| 2.2      | API Reference . . . . .        | 4         |
| <b>3</b> | <b>configconfig.configvar</b>  | <b>5</b>  |
| 3.1      | ConfigVar . . . . .            | 5         |
| <b>4</b> | <b>configconfig.metaclass</b>  | <b>7</b>  |
| 4.1      | ConfigVarMeta . . . . .        | 7         |
| <b>5</b> | <b>configconfig.parser</b>     | <b>9</b>  |
| 5.1      | Parser . . . . .               | 9         |
| <b>6</b> | <b>configconfig.testing</b>    | <b>11</b> |
| 6.1      | ConfigVarTest . . . . .        | 11        |
| 6.2      | NotIntTest . . . . .           | 12        |
| 6.3      | NotBoolTest . . . . .          | 12        |
| 6.4      | NotStrTest . . . . .           | 12        |
| 6.5      | ListTest . . . . .             | 12        |
| 6.6      | DirectoryTest . . . . .        | 13        |
| 6.7      | BoolTrueTest . . . . .         | 14        |
| 6.8      | BoolFalseTest . . . . .        | 15        |
| 6.9      | RequiredStringTest . . . . .   | 15        |
| 6.10     | OptionalStringTest . . . . .   | 16        |
| 6.11     | EnumTest . . . . .             | 17        |
| 6.12     | DictTest . . . . .             | 17        |
| <b>7</b> | <b>configconfig.utils</b>      | <b>19</b> |
| 7.1      | get_literal_values . . . . .   | 19        |
| 7.2      | optional_getter . . . . .      | 19        |
| 7.3      | get_yaml_type . . . . .        | 19        |
| 7.4      | make_schema . . . . .          | 20        |
| 7.5      | check_union . . . . .          | 20        |
| 7.6      | get_json_type . . . . .        | 20        |
| <b>8</b> | <b>configconfig.validator</b>  | <b>21</b> |
| 8.1      | Validator . . . . .            | 21        |
| 8.2      | validate_files . . . . .       | 23        |

|                            |           |
|----------------------------|-----------|
| <b>Python Module Index</b> | <b>25</b> |
| <b>Index</b>               | <b>27</b> |

## Installation

### 1.1 from PyPI

```
$ python3 -m pip install configconfig --user
```

### 1.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge  
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install configconfig
```

### 1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/repo-helper/configconfig@master --user
```



## configconfig.autoconfig

A Sphinx directive for documenting YAML configuration values.

Provides the `autoconfig` directive to document configuration values automatically, the `conf` directive to document them manually, and the `conf` role to link to a `conf` directive.

**Attention:** This module has the following additional requirements:

```
docutils
sphinx<3.4.0,>=3.0.3
sphinx-toolbox
```

These can be installed as follows:

```
$ python -m pip install configconfig[sphinx]
```

## 2.1 Usage

### **.. autoconfig::**

Directive to automatically document an YAML configuration value.

Takes a single argument, either the fully qualified name of the `ConfigVar` object, or the name of the module if the `:category:` option is given.

**:category: (string)**

(optional) The category of options to document.

### **.. conf::**

Directive to document an YAML configuration value.

### **:conf:**

Role to add a cross-reference to a `conf` or `autoconfig` directive.

## 2.2 API Reference

### Classes:

|  |   |
|--|---|
| <code>AutoConfigDirective(name, arguments, ...)</code> | Sphinx directive to automatically document an YAML configuration value. |
|--|---|

### Functions:

|   |   |
|---|---|
| <code>parse_conf_node(env, text, node)</code> | Parse the content of a <code>conf</code> directive. |
| <code>setup(app)</code>                       | Setup Sphinx Extension.                             |

**class** `AutoConfigDirective` (*name, arguments, options, content, lineno, content\_offset, block\_text, state, state\_machine*)

Bases: `SphinxDirective`

Sphinx directive to automatically document an YAML configuration value.

### Methods:

|   |   |
|---|---|
| <code>document_config_var(var_obj)</code> | Document the given configuration value. |
| <code>run()</code>                        | Process the content of the directive.   |

**document\_config\_var** (*var\_obj*)  
Document the given configuration value.

**Parameters** `var_obj` (`Type[ConfigVar]`)

**Return type** `paragraph`

**run** ()  
Process the content of the directive.

**Return type** `Sequence[Node]`

**parse\_conf\_node** (*env, text, node*)  
Parse the content of a `conf` directive.

### Parameters

- **env** (`BuildEnvironment`) – The Sphinx build environment.
- **text** (`str`) – The content of the directive.
- **node** (`desc_signature`) – The docutils node class.

**Return type** `str`

**setup** (*app*)  
Setup Sphinx Extension.

**Parameters** `app` (`Sphinx`) – The Sphinx app

**Return type** `Dict[str, Any]`



## configconfig.configvar

**class** `ConfigVar`(*raw\_config\_vars: Dict[str, Any]*)

Base class for YAML configuration values.

The class docstring should be the description of the config var, with an example, and the name of the class should be the variable name.

Alternatively, for a more Pythonic naming approach, the variable name can be set with the `name` class variable.

### Example:

```
class platforms(ConfigVar):
    """
    A case-insensitive list of platforms to perform tests for.

    Example:

    .. code-block:: yaml

        platforms:
            - Windows
            - Linux

    These values determine the GitHub test workflows to enable,
    and the Trove classifiers used on PyPI.
    """

    dtype = List[Literal["Windows", "macOS", "Linux"]]
    default: List[str] = ["Windows", "macOS", "Linux"]
    category: str = "packaging"
```

### Attributes:

|                       |   |
|-----------------------|---|
| <code>category</code> | The category the <code>ConfigVar</code> is listed under in the documentation. |
| <code>default</code>  | The default value of the configuration value if it is optional.               |
| <code>dtype</code>    | The allowed type or types in the YAML configuration file.                     |
| <code>required</code> | Flag to indicate whether the configuration value is required.                 |
| <code>rtype</code>    | The variable type passed to Jinja2.   |

### Methods:

|  |   |
|--|---|
| <code>get([raw_config_vars])</code>      | Returns the value of this <code>ConfigVar</code> .  |
| <code>make_documentation()</code>        | Returns the reStructuredText documentation for the <code>ConfigVar</code> .                 |
| <code>validate([raw_config_vars])</code> | Validate the value obtained from the YAML file and coerce into the appropriate return type. |
| <code>validator(value)</code>            | Function to call to validate the values.  |

**category** = 'other'

**Type:** `str`

The category the `ConfigVar` is listed under in the documentation.

**default** = ''

**Type:** `Union[Callable[[Dict[str, Any]], Any], Any]`

The default value of the configuration value if it is optional. Defaults to '' if unset.

May also be set to a callable which returns a dynamic or mutable default value.

**dtype** = `typing.Any`

**Type:** `Type`

The allowed type or types in the YAML configuration file.

**classmethod get** (*raw\_config\_vars=None*)

Returns the value of this `ConfigVar`.

**Parameters** **raw\_config\_vars** (`Optional[Dict[str, Any]]`) – Dictionary to obtain the value from. Default `None`.

**Return type** See the `rtype` attribute.

**classmethod make\_documentation** ()

Returns the reStructuredText documentation for the `ConfigVar`.

**Return type** `str`

**required** = `False`

**Type:** `bool`

Flag to indicate whether the configuration value is required. Default `False`.

**rtype** = `typing.Any`

**Type:** `Type`

The variable type passed to Jinja2. If `None` `dtype` is used. Ignored for `dtype=bool`.

**classmethod validate** (*raw\_config\_vars=None*)

Validate the value obtained from the YAML file and coerce into the appropriate return type.

**Parameters** **raw\_config\_vars** (`Optional[Dict[str, Any]]`) – Dictionary to obtain the value from. Default `None`.

**Return type** See the `rtype` attribute.

**classmethod validator** (*value*)

Function to call to validate the values.

- The callable must have a single required argument (the value).
- Should raise `ValueError` if values are invalid, and return the values if they are valid.
- May change the values (e.g. make lowercase) before returning.

**Return type** `Any`

## configconfig.metaclass

**class** **ConfigVarMeta** (*name: str, bases, dct: Dict*)

Bases: `type`

Metaclass for configuration values.

### Methods:

|   |   |
|---|---|
| <code>__call__(raw_config_vars)</code>  | Alias for <code>ConfigVar.get</code> .                                  |
| <code>__repr__()</code>                 | Return a string representation of the <code>ConfigVarMeta</code> class. |
| <code>get_schema_entry([schema])</code> | Returns the JSON schema entry for this configuration value.             |

`__call__(raw_config_vars)`

Alias for `ConfigVar.get`.

Returns the value of the `ConfigVar`.

**Parameters** `raw_config_vars` (`Dict[str, Any]`) – Dictionary to obtain the value from.

**Return type** See the `ConfigVar.rtype` attribute.

`__repr__()`

Return a string representation of the `ConfigVarMeta` class.

**Return type** `str`

`get_schema_entry(schema=None)`

Returns the JSON schema entry for this configuration value.

**Parameters** `schema` (`Optional[Dict]`) – Default `None`.

**Return type** `Dict[str, Any]`

**Returns** Dictionary representation of the JSON schema.



## configconfig.parser

**class Parser** (*allow\_unknown\_keys=False*)  
Base class for YAML configuration parsers.

Custom parsing steps for each configuration variable can be implemented with methods in the form:

```
def visit_<configuration value name>(
    self,
    raw_config_vars: Dict[str, Any],
) -> Any: ...
```

The method must return the value to set the configuration variable to, or raise an error in the case of invalid input.

A final custom parsing step, useful when several values must be set at once, may be implemented in the `custom_parsing` method:

```
def custom_parsing(
    self,
    raw_config_vars: Mapping[str, Any],
    parsed_config_vars: MutableMapping[str, Any],
    filename: PathPlus,
) -> MutableMapping[str, Any]: ...
```

This takes the mapping of raw configuration variables, the mapping of parsed variables (those set with the `visit_<configuration value name>` method), and the configuration file name. The method must return the `parsed_config_vars`.

### Methods:

|   |  |
|---|--|
| <code>custom_parsing(raw_config_vars, ...)</code> | Custom parsing step.                     |
| <code>run(filename)</code>                        | Parse configuration from the given file. |

**custom\_parsing** (*raw\_config\_vars, parsed\_config\_vars, filename*)  
Custom parsing step.

### Parameters

- **raw\_config\_vars** (`Mapping[str, Any]`) – Mapping of raw configuration values loaded from the YAML configuration file.
- **parsed\_config\_vars** (`MutableMapping[str, Any]`) – Mapping of parsed configuration values.
- **filename** (`PathPlus`) – The filename of the YAML configuration file.

**Return type** `MutableMapping[str, Any]`

**run** (*filename*)

Parse configuration from the given file.

**Parameters** **filename** (`Union[str, Path, PathLike]`) – The filename of the YAML configuration file.

**Return type** `MutableMapping[str, Any]`

## configconfig.testing

Helpers for testing *ConfigVar*.

**Attention:** This module has the following additional requirement:

```
pytest
```

This can be installed as follows:

```
$ python -m pip install configconfig[testing]
```

New in version 0.2.0.

### Classes:

|                             |  |
|-----------------------------|--|
| <i>BoolFalseTest()</i>      | Test for boolean configuration values which default to <i>False</i> .  |
| <i>BoolTrueTest()</i>       | Test for boolean configuration values which default to <i>True</i> .   |
| <i>ConfigVarTest()</i>      | Base class for tests of <i>ConfigVars</i> .                            |
| <i>DictTest()</i>           | Test for dictionary configuration values.                              |
| <i>DirectoryTest()</i>      | Test for configuration values which represent directories.             |
| <i>EnumTest()</i>           | Test for <i>Enum</i> configuration values.                             |
| <i>ListTest()</i>           | Test for list configuration values.                                    |
| <i>NotBoolTest()</i>        | Mixin to add tests for <i>ConfigVars</i> that can't be boolean values. |
| <i>NotIntTest()</i>         | Mixin to add tests for <i>ConfigVars</i> that can't be integers.       |
| <i>NotStrTest()</i>         | Mixin to add tests for <i>ConfigVars</i> that can't be strings.        |
| <i>OptionalStringTest()</i> | Test for string configuration values which are optional.               |
| <i>RequiredStringTest()</i> | Test for string configuration values which are required.               |

### class ConfigVarTest

Bases: ABC

Base class for tests of *ConfigVars*.

#### config\_var

**Type:** `Type[ConfigVar]`

The *ConfigVar* under test.

**class NotIntTest**Bases: *ConfigVarTest*Mixin to add tests for *ConfigVars* that can't be integers.**test\_error\_int()**Checks that the *ConfigVar* raises a *ValueError* when passed an *int*.**class NotBoolTest**Bases: *ConfigVarTest*Mixin to add tests for *ConfigVars* that can't be boolean values.**test\_error\_bool()**Checks that the *ConfigVar* raises a *ValueError* when passed a *bool*.**class NotStrTest**Bases: *ConfigVarTest*Mixin to add tests for *ConfigVars* that can't be strings.**test\_error\_str()**Checks that the *ConfigVar* raises a *ValueError* when passed a *str*.**class ListTest**Bases: *NotStrTest*, *NotBoolTest*, *NotIntTest*, *ConfigVarTest*

Test for list configuration values.

**Attributes:**

|                            |   |
|----------------------------|---|
| <i>default_value</i>       | The default value that should be returned when no valid is given.                           |
| <i>different_key_value</i> | A dictionary containing one or more keys that are not the keys used by the <i>ConfigVar</i> |
| <i>test_value</i>          | A value that is valid and should be returned unchanged.                                     |

**Methods:**

|                       |  |
|-----------------------|--|
| <i>test_success()</i> | Checks that the <i>ConfigVar</i> can correctly parse various <i>list</i> values. |
|-----------------------|--|

**default\_value = []****Type:** *List[str]*

The default value that should be returned when no valid is given.

**different\_key\_value = {'username': 'domdfcoding'}****Type:** *Dict[str, Any]*A dictionary containing one or more keys that are not the keys used by the *ConfigVar***test\_success()**Checks that the *ConfigVar* can correctly parse various *list* values.



**test\_value**

**Type:** `List[str]`

A value that is valid and should be returned unchanged.

**class DirectoryTest**

Bases: `NotBoolTest`, `NotIntTest`, `ConfigVarTest`

Test for configuration values which represent directories.

**Attributes:**

|                                  |   |
|----------------------------------|---|
| <code>default_value</code>       | The default value that should be returned when no valid is given.                                 |
| <code>different_key_value</code> | A dictionary containing one or more keys that are not the keys used by the <code>ConfigVar</code> |
| <code>test_value</code>          | A value that is valid and should be returned unchanged.   |

**Methods:**

|                                    |  |
|------------------------------------|--|
| <code>test_error_list_int()</code> | Checks that the <code>ConfigVar</code> raises a <code>ValueError</code> when passed a <code>str</code> . |
| <code>test_error_list_str()</code> | Checks that the <code>ConfigVar</code> raises a <code>ValueError</code> when passed a <code>str</code> . |
| <code>test_success()</code>        | Checks that the <code>ConfigVar</code> can correctly parse various directory values.                     |

**default\_value**

**Type:** `str`

The default value that should be returned when no valid is given.

**different\_key\_value = {'username': 'domdfcoding'}**

**Type:** `Dict[str, Any]`

A dictionary containing one or more keys that are not the keys used by the `ConfigVar`

**test\_error\_list\_int()**

Checks that the `ConfigVar` raises a `ValueError` when passed a `str`.

**test\_error\_list\_str()**

Checks that the `ConfigVar` raises a `ValueError` when passed a `str`.

**test\_success()**

Checks that the `ConfigVar` can correctly parse various directory values.

**test\_value**

**Type:** `str`

A value that is valid and should be returned unchanged.

**class BoolTrueTest**Bases: *ConfigVarTest*Test for boolean configuration values which default to *True*.**Attributes:**

|                            |   |
|----------------------------|---|
| <i>different_key_value</i> | A dictionary containing one or more keys that are not the keys used by the <i>ConfigVar</i> |
| <i>false_values</i>        | A list of values which should be considered <i>False</i> by the <i>ConfigVar</i> .          |
| <i>true_values</i>         | A list of values which should be considered <i>True</i> by the <i>ConfigVar</i> .           |
| <i>wrong_values</i>        | A list of values which should are of the wrong type.  |

**Methods:**

|                         |
|-------------------------|
| <i>test_empty_get()</i> |
| <i>test_errors()</i>    |
| <i>test_false()</i>     |
| <i>test_true()</i>      |

**different\_key\_value** = {'username': 'domdfcoding'}**Type:** *Dict[str, Any]*A dictionary containing one or more keys that are not the keys used by the *ConfigVar***property false\_values**A list of values which should be considered *False* by the *ConfigVar*.**Return type** *List[Dict[str, Any]]***test\_empty\_get()****test\_errors()****test\_false()****test\_true()****property true\_values**A list of values which should be considered *True* by the *ConfigVar*.**Return type** *List[Dict[str, Any]]***property wrong\_values**

A list of values which should are of the wrong type.

**Return type** *List[Dict[str, Any]]*

**class BoolFalseTest**Bases: *BoolTrueTest*Test for boolean configuration values which default to *False*.**Attributes:**

|                            |   |
|----------------------------|---|
| <i>different_key_value</i> | A dictionary containing one or more keys that are not the keys used by the <i>ConfigVar</i> |
| <i>false_values</i>        | A list of values which should be considered <i>False</i> by the <i>ConfigVar</i> .          |
| <i>true_values</i>         | A list of values which should be considered <i>True</i> by the <i>ConfigVar</i> .           |

**Methods:**

|                         |
|-------------------------|
| <i>test_empty_get()</i> |
|-------------------------|

**different\_key\_value** = {'username': 'domdfcoding'}**Type:** *Dict[str, Any]*A dictionary containing one or more keys that are not the keys used by the *ConfigVar***property false\_values**A list of values which should be considered *False* by the *ConfigVar*.**Return type** *List[Dict[str, Any]]***test\_empty\_get()****property true\_values**A list of values which should be considered *True* by the *ConfigVar*.**Return type** *List[Dict[str, Any]]***class RequiredStringTest**Bases: *ConfigVarTest*

Test for string configuration values which are required.

**Methods:**

|                         |
|-------------------------|
| <i>test_empty_get()</i> |
| <i>test_errors()</i>    |
| <i>test_success()</i>   |

**Attributes:**

|                     |   |
|---------------------|---|
| <i>test_value</i>   | A value that is valid and should be returned unchanged. |
| <i>wrong_values</i> | A list of values which should are of the wrong type.    |

`test_empty_get()`

`test_errors()`

`test_success()`

`test_value`

**Type:** `str`

A value that is valid and should be returned unchanged.

**property wrong\_values**

A list of values which should are of the wrong type.

**Return type** `List[Dict[str, Any]]`

**class OptionalStringTest**

Bases: `RequiredStringTest`

Test for string configuration values which are optional.

**Attributes:**

|                                  |   |
|----------------------------------|---|
| <code>default_value</code>       | The default value that should be returned when no valid is given.                                 |
| <code>different_key_value</code> | A dictionary containing one or more keys that are not the keys used by the <code>ConfigVar</code> |
| <code>wrong_values</code>        | A list of values which should are of the wrong type.  |

**Methods:**

|                               |
|-------------------------------|
| <code>test_empty_get()</code> |
| <code>test_errors()</code>    |
| <code>test_success()</code>   |

`default_value = ''`

**Type:** `str`

The default value that should be returned when no valid is given.

`different_key_value = {'sphinx_html_theme': 'alabaster'}`

**Type:** `Dict[str, Any]`

A dictionary containing one or more keys that are not the keys used by the `ConfigVar`

`test_empty_get()`

`test_errors()`

`test_success()`

**property wrong\_values**

A list of values which should are of the wrong type.

**Return type** `List[Dict[str, Any]]`

**class EnumTest**

Bases: *RequiredStringTest*

Test for `Enum` configuration values.

**Attributes:**

|                              |   |
|------------------------------|---|
| <code>default_value</code>   | The default value that should be returned when no valid is given. |
| <code>non_enum_values</code> | A list of values which are of the correct type but are invalid.   |

**Methods:**

|                               |
|-------------------------------|
| <code>test_empty_get()</code> |
| <code>test_errors()</code>    |
| <code>test_non_enum()</code>  |

**default\_value**

**Type:** `str`

The default value that should be returned when no valid is given.

**non\_enum\_values**

**Type:** `List[Any]`

A list of values which are of the correct type but are invalid.

**test\_empty\_get ()**

**test\_errors ()**

**test\_non\_enum ()**

**class DictTest**

Bases: *NotStrTest, NotBoolTest, NotIntTest, ConfigVarTest*

Test for dictionary configuration values.

**Attributes:**

|                                  |   |
|----------------------------------|---|
| <code>default_value</code>       | The default value that should be returned when no valid is given.                           |
| <code>different_key_value</code> | A dictionary containing one or more keys that are not the keys used by the <i>ConfigVar</i> |
| <code>test_value</code>          | A value that is valid and should be returned unchanged.                                     |

**Methods:**

|                                    |
|------------------------------------|
| <code>test_error_list_int()</code> |
| <code>test_success()</code>        |

**default\_value = {}**

**Type:** `Dict[str, Any]`

The default value that should be returned when no valid is given.

**different\_key\_value = {'sphinx\_html\_theme': 'alabaster'}**

**Type:** `Dict[str, Any]`

A dictionary containing one or more keys that are not the keys used by the *ConfigVar*

**test\_error\_list\_int()**

**test\_success()**

**test\_value**

**Type:** `Dict[str, Any]`

A value that is valid and should be returned unchanged.

## configconfig.utils

Utility functions.

### Functions:

|  |  |
|--|--|
| <code>get_literal_values(literal)</code>                     | Returns a tuple of permitted values for a <code>typing.Literal</code> .  |
| <code>optional_getter(raw_config_vars, cls, required)</code> | Returns either the configuration value, the default, or raises an error if the value is required but wasn't supplied.    |
| <code>get_yaml_type(type_)</code>                            | Get the YAML type that corresponds to the given Python type.   |
| <code>make_schema(*configuration_variables)</code>           | Create a JSON schema from a list of <code>ConfigVar</code> classes.  |
| <code>check_union(obj, dtype)</code>                         | Check if the type of <code>obj</code> is one of the types in a <code>typing.Union</code> , <code>typing.List</code> etc. |
| <code>get_json_type(type_)</code>                            | Get the type for the JSON schema that corresponds to the given Python type.  |

#### **get\_literal\_values** (*literal*)

Returns a tuple of permitted values for a `typing.Literal`.

New in version 0.3.0.

**Parameters** `literal` (`Literal[]`)

**Return type** `Tuple[Any]`

#### **optional\_getter** (*raw\_config\_vars, cls, required*)

Returns either the configuration value, the default, or raises an error if the value is required but wasn't supplied.

##### **Parameters**

- **raw\_config\_vars** (`Dict[str, Any]`)
- **cls** (`ConfigVarMeta`)
- **required** (`bool`)

**Return type** `Any`

#### **get\_yaml\_type** (*type\_*)

Get the YAML type that corresponds to the given Python type.

**Parameters** `type_` (`Type`)

**Return type** `str`

**make\_schema** (\**configuration\_variables*)

Create a JSON schema from a list of *ConfigVar* classes.

**Parameters** *configuration\_variables*

**Return type** `Dict[str, Any]`

**Returns** Dictionary representation of the JSON schema.

**check\_union** (*obj*, *dtype*)

Check if the type of *obj* is one of the types in a `typing.Union`, `typing.List` etc.

**Parameters**

- *obj* (*Any*)
- *dtype* (`Union`, `List`, etc.)

**Return type** `bool`

**get\_json\_type** (*type\_*)

Get the type for the JSON schema that corresponds to the given Python type.

**Parameters** *type\_* (*Type*)

**Return type** `Dict[str, Union[str, List, Dict]]`



## **configconfig.validator**

Validate values obtained from the YAML file and coerce into the appropriate return types.

### **Classes:**

---

|                                    |   |
|------------------------------------|---|
| <code>Validator(config_var)</code> | Methods are named <code>visit_&lt;type&gt;</code> . |
|------------------------------------|---|

---

### **Functions:**

---

|  |  |
|--|--|
| <code>validate_files(schemafile, *datafiles[, ...])</code> | Validate the given datafiles against the given schema. |
|--|--|

---

**class Validator** (*config\_var*)

Bases: `object`

Methods are named `visit_<type>`.

### **Methods:**

---

|   |  |
|---|--|
| <code>unknown_type()</code>                 | Called when the desired type has no visitor.                     |
| <code>validate([raw_config_vars])</code>    | Validate the configuration value.                                |
| <code>visit_bool(raw_config_vars)</code>    | Used to validate and convert <code>bool</code> values.           |
| <code>visit_dict(raw_config_vars)</code>    | Used to validate and convert <code>dict</code> values.           |
| <code>visit_float(raw_config_vars)</code>   | Used to validate and convert <code>float</code> values.          |
| <code>visit_int(raw_config_vars)</code>     | Used to validate and convert <code>int</code> values.            |
| <code>visit_list(raw_config_vars)</code>    | Used to validate and convert <code>list</code> values.           |
| <code>visit_literal(raw_config_vars)</code> | Used to validate and convert <code>typing.Literal</code> values. |
| <code>visit_str(raw_config_vars)</code>     | Used to validate and convert <code>str</code> values.            |
| <code>visit_union(raw_config_vars)</code>   | Used to validate and convert <code>typing.Union</code> values.   |

---

**unknown\_type** ()

Called when the desired type has no visitor.

**Return type** `NoReturn`

**validate** (*raw\_config\_vars=None*)

Validate the configuration value.

**Parameters** `raw_config_vars` (`Optional[Dict[str, Any]]`) – Default `None`.

**Return type** `Any`

**Returns** The validated value.

**visit\_bool** (*raw\_config\_vars*)

Used to validate and convert `bool` values.

**Parameters** `raw_config_vars` (`Dict[str, Any]`)

**Return type** `bool`

**visit\_dict** (*raw\_config\_vars*)

Used to validate and convert `dict` values.

**Parameters** `raw_config_vars` (`Dict[str, Any]`)

**Return type** `Dict`

**visit\_float** (*raw\_config\_vars*)

Used to validate and convert `float` values.

**Parameters** `raw_config_vars` (`Dict[str, Any]`)

**Return type** `float`

**visit\_int** (*raw\_config\_vars*)

Used to validate and convert `int` values.

**Parameters** `raw_config_vars` (`Dict[str, Any]`)

**Return type** `int`

**visit\_list** (*raw\_config\_vars*)

Used to validate and convert `list` values.

**Parameters** `raw_config_vars` (`Dict[str, Any]`)

**Return type** `List`

**visit\_literal** (*raw\_config\_vars*)

Used to validate and convert `typing.Literal` values.

**Parameters** `raw_config_vars` (`Dict[str, Any]`)

**Return type** `Any`

**visit\_str** (*raw\_config\_vars*)

Used to validate and convert `str` values.

**Parameters** `raw_config_vars` (`Dict[str, Any]`)

**Return type** `str`

**visit\_union** (*raw\_config\_vars*)

Used to validate and convert `typing.Union` values.

**Parameters** `raw_config_vars` (`Dict[str, Any]`)

**Return type** `Any`

**validate\_files** (*schemafile*, \**datafiles*, *encoding*='utf-8')

Validate the given datafiles against the given schema.

**Parameters**

- **schemafile** (`Union[str, Path, PathLike]`) – The json or yaml formatted schema to validate with.
- **\*datafiles** (`Union[str, Path, PathLike]`) – The json or yaml files to validate.
- **encoding** (`str`) – Encoding to open the files with. Default 'utf-8'.

New in version 0.4.0.



## Python Module Index

### C

- `configconfig.autoconfig`, [3](#)
- `configconfig.configvar`, [5](#)
- `configconfig.metaclass`, [7](#)
- `configconfig.parser`, [9](#)
- `configconfig.testing`, [11](#)
- `configconfig.utils`, [19](#)
- `configconfig.validator`, [21](#)



## Symbols

`:category:` (*directive option*)  
     `autoconfig` (*directive*), 3  
`__call__()` (*ConfigVarMeta method*), 7  
`__repr__()` (*ConfigVarMeta method*), 7

## A

`autoconfig` (*directive*), 3  
     `:category:` (*directive option*), 3  
`AutoConfigDirective` (*class in configconfig.autoconfig*), 4

## B

`BoolFalseTest` (*class in configconfig.testing*), 15  
`BoolTrueTest` (*class in configconfig.testing*), 14

## C

`category` (*ConfigVar attribute*), 5  
`check_union()` (*in module configconfig.utils*), 20  
`conf` (*directive*), 3  
`conf` (*role*), 3  
`config_var` (*ConfigVarTest attribute*), 11  
`configconfig.autoconfig`  
     *module*, 3  
`configconfig.configvar`  
     *module*, 5  
`configconfig.metaclass`  
     *module*, 7  
`configconfig.parser`  
     *module*, 9  
`configconfig.testing`  
     *module*, 11  
`configconfig.utils`  
     *module*, 19  
`configconfig.validator`  
     *module*, 21  
`ConfigVar` (*class in configconfig.configvar*), 5  
`ConfigVarMeta` (*class in configconfig.metaclass*), 7  
`ConfigVarTest` (*class in configconfig.testing*), 11  
`custom_parsing()` (*Parser method*), 9

## D

`default` (*ConfigVar attribute*), 6

`default_value` (*DictTest attribute*), 17  
`default_value` (*DirectoryTest attribute*), 13  
`default_value` (*EnumTest attribute*), 17  
`default_value` (*ListTest attribute*), 12  
`default_value` (*OptionalStringTest attribute*), 16  
`DictTest` (*class in configconfig.testing*), 17  
`different_key_value` (*BoolFalseTest attribute*),  
     15  
`different_key_value` (*BoolTrueTest attribute*), 14  
`different_key_value` (*DictTest attribute*), 18  
`different_key_value` (*DirectoryTest attribute*), 13  
`different_key_value` (*ListTest attribute*), 12  
`different_key_value` (*OptionalStringTest*  
     *attribute*), 16  
`DirectoryTest` (*class in configconfig.testing*), 13  
`document_config_var()` (*AutoConfigDirective*  
     *method*), 4  
`dtype` (*ConfigVar attribute*), 6

## E

`EnumTest` (*class in configconfig.testing*), 17

## F

`false_values()` (*BoolFalseTest property*), 15  
`false_values()` (*BoolTrueTest property*), 14

## G

`get()` (*ConfigVar class method*), 6  
`get_json_type()` (*in module configconfig.utils*), 20  
`get_literal_values()` (*in module*  
     *configconfig.utils*), 19  
`get_schema_entry()` (*ConfigVarMeta method*), 7  
`get_yaml_type()` (*in module configconfig.utils*), 19

## L

`ListTest` (*class in configconfig.testing*), 12

## M

`make_documentation()` (*ConfigVar class method*),  
     6  
`make_schema()` (*in module configconfig.utils*), 19  
*module*  
     `configconfig.autoconfig`, 3

configconfig.configvar, 5  
 configconfig.metaclass, 7  
 configconfig.parser, 9  
 configconfig.testing, 11  
 configconfig.utils, 19  
 configconfig.validator, 21

## N

non\_enum\_values (*EnumTest attribute*), 17  
 NotBoolTest (*class in configconfig.testing*), 12  
 NotIntTest (*class in configconfig.testing*), 12  
 NotStrTest (*class in configconfig.testing*), 12

## O

optional\_getter() (*in module configconfig.utils*), 19  
 OptionalStringTest (*class in configconfig.testing*), 16

## P

parse\_conf\_node() (*in module configconfig.autoconfig*), 4  
 Parser (*class in configconfig.parser*), 9

## R

required (*ConfigVar attribute*), 6  
 RequiredStringTest (*class in configconfig.testing*), 15  
 rtype (*ConfigVar attribute*), 6  
 run() (*AutoConfigDirective method*), 4  
 run() (*Parser method*), 9

## S

setup() (*in module configconfig.autoconfig*), 4

## T

test\_empty\_get() (*BoolFalseTest method*), 15  
 test\_empty\_get() (*BoolTrueTest method*), 14  
 test\_empty\_get() (*EnumTest method*), 17  
 test\_empty\_get() (*OptionalStringTest method*), 16  
 test\_empty\_get() (*RequiredStringTest method*), 15  
 test\_error\_bool() (*NotBoolTest method*), 12  
 test\_error\_int() (*NotIntTest method*), 12  
 test\_error\_list\_int() (*DictTest method*), 18  
 test\_error\_list\_int() (*DirectoryTest method*), 13  
 test\_error\_list\_str() (*DirectoryTest method*), 13  
 test\_error\_str() (*NotStrTest method*), 12  
 test\_errors() (*BoolTrueTest method*), 14  
 test\_errors() (*EnumTest method*), 17  
 test\_errors() (*OptionalStringTest method*), 16  
 test\_errors() (*RequiredStringTest method*), 16

test\_false() (*BoolTrueTest method*), 14  
 test\_non\_enum() (*EnumTest method*), 17  
 test\_success() (*DictTest method*), 18  
 test\_success() (*DirectoryTest method*), 13  
 test\_success() (*ListTest method*), 12  
 test\_success() (*OptionalStringTest method*), 16  
 test\_success() (*RequiredStringTest method*), 16  
 test\_true() (*BoolTrueTest method*), 14  
 test\_value (*DictTest attribute*), 18  
 test\_value (*DirectoryTest attribute*), 13  
 test\_value (*ListTest attribute*), 12  
 test\_value (*RequiredStringTest attribute*), 16  
 true\_values() (*BoolFalseTest property*), 15  
 true\_values() (*BoolTrueTest property*), 14

## U

unknown\_type() (*Validator method*), 21

## V

validate() (*ConfigVar class method*), 6  
 validate() (*Validator method*), 21  
 validate\_files() (*in module configconfig.validator*), 22  
 Validator (*class in configconfig.validator*), 21  
 validator() (*ConfigVar class method*), 6  
 visit\_bool() (*Validator method*), 21  
 visit\_dict() (*Validator method*), 22  
 visit\_float() (*Validator method*), 22  
 visit\_int() (*Validator method*), 22  
 visit\_list() (*Validator method*), 22  
 visit\_literal() (*Validator method*), 22  
 visit\_str() (*Validator method*), 22  
 visit\_union() (*Validator method*), 22

## W

wrong\_values() (*BoolTrueTest property*), 14  
 wrong\_values() (*OptionalStringTest property*), 16  
 wrong\_values() (*RequiredStringTest property*), 16