coala Documentation

Release 0.9.0

The coala Developers

Nov 22, 2016

Contents

Hey there! You're in the right place if you:

- want to develop coala itself!
- want to develop a bear for coala.

If you're trying to **use** coala, you should have a look at our user documentation instead.

CHAPTER 1

coalib package

1.1 Subpackages

1.1.1 coalib.bearlib package

Subpackages

coalib.bearlib.abstractions package

Submodules

coalib.bearlib.abstractions.ExternalBearWrap module

coalib.bearlib.abstractions.ExternalBearWrap.**external_bear_wrap**(*executable: str*, ***options*)

coalib.bearlib.abstractions.Linter module

coalib.bearlib.abstractions.Linter.**linter**(*executable: str*, *use_stdin: bool = False*, *use_stdout: bool = True*, *use_stderr: bool = False*, *config_suffix: str = ''*, *executable_check_fail_info: str = ''*, *prerequisite_check_command: tuple = ()*, *output_format: (<class 'str'>, None) = None*, ***options*)

Decorator that creates a `LocalBear` that is able to process results from an external linter tool.

The main functionality is achieved through the `create_arguments()` function that constructs the command-line-arguments that get parsed to your executable.

```
>>> @linter("xlint", output_format="regex", output_regex="...")
... class XLintBear:
...     @staticmethod
...     def create_arguments(filename, file, config_file):
...         return "--lint", filename
```

Requiring settings is possible like in `Bear.run()` with supplying additional keyword arguments (and if needed with defaults).

```
>>> @linter("xlint", output_format="regex", output_regex="...")
... class XLintBear:
...     @staticmethod
...     def create_arguments(filename,
...                          file,
...                          config_file,
...                          lintmode: str,
...                          enable_aggressive_lints: bool=False):
...         arguments = ("--lint", filename, "--mode=" + lintmode)
...         if enable_aggressive_lints:
...             arguments += ("--aggressive",)
...         return arguments
```

Sometimes your tool requires an actual file that contains configuration. `linter` allows you to just define the contents the configuration shall contain via `generate_config()` and handles everything else for you.

```
>>> @linter("xlint", output_format="regex", output_regex="...")
... class XLintBear:
...     @staticmethod
...     def generate_config(filename,
...                         file,
...                         lintmode,
...                         enable_aggressive_lints):
...         modestring = ("aggressive"
...                       if enable_aggressive_lints else
...                       "non-aggressive")
...         contents = ("<xlint>",
...                     "    <mode>" + lintmode + "</mode>",
...                     "    <aggressive>" + modestring + "</aggressive>",
...                     "</xlint>")
...         return "\n".join(contents)
...
...     @staticmethod
...     def create_arguments(filename,
...                          file,
...                          config_file):
...         return "--lint", filename, "--config", config_file
```

As you can see you don't need to copy additional keyword-arguments you introduced from `create_arguments()` to `generate_config()` and vice-versa. `linter` takes care of forwarding the right arguments to the right place, so you are able to avoid signature duplication.

If you override `process_output`, you have the same feature like above (auto-forwarding of the right arguments defined in your function signature).

Note when overriding `process_output`: Providing a single output stream (via `use_stdout` or `use_stderr`) puts the according string attained from the stream into parameter `output`, providing both output streams inputs a tuple with `(stdout, stderr)`. Providing `use_stdout=False` and `use_stderr=False` raises a `ValueError`. By default `use_stdout` is `True` and `use_stderr` is `False`.

Documentation: Bear description shall be provided at class level. If you document your additional parameters inside `create_arguments`, `generate_config` and `process_output`, beware that conflicting documentation between them may be overridden. Document duplicated parameters inside `create_arguments` first, then in `generate_config` and after that inside `process_output`.

For the tutorial see: http://coala.readthedocs.io/en/latest/Developers/Writing_Linter_Bears.html

Parameters

- **executable** – The linter tool.
- **use_stdin** – Whether the input file is sent via stdin instead of passing it over the command-line-interface.
- **use_stdout** – Whether to use the stdout output stream.
- **use_stderr** – Whether to use the stderr output stream.
- **config_suffix** – The suffix-string to append to the filename of the configuration file created when `generate_config` is supplied. Useful if your executable expects getting a specific file-type with specific file-ending for the configuration file.
- **executable_check_fail_info** – Information that is provided together with the fail message from the normal executable check. By default no additional info is printed.
- **prerequisite_check_command** – A custom command to check for when `check_prerequisites` gets invoked (via `subprocess.check_call()`). Must be an `Iterable`.
- **prerequisite_check_fail_message** – A custom message that gets displayed when `check_prerequisites` fails while invoking `prerequisite_check_command`. Can only be provided together with `prerequisite_check_command`.
- **output_format** – The output format of the underlying executable. Valid values are
  - `None`: Define your own format by overriding `process_output`. Overriding `process_output` is then mandatory, not specifying it raises a `ValueError`.
  - `'regex'`: Parse output using a regex. See parameter `output_regex`.
  - `'corrected'`: The output is the corrected of the given file. Diffs are then generated to supply patches for results.

  Passing something else raises a `ValueError`.
- **output_regex** – The regex expression as a string that is used to parse the output generated by the underlying executable. It should use as many of the following named groups (via `(?P<name>...)`) to provide a good result:
  - line - The line where the issue starts.
  - column - The column where the issue starts.
  - end_line - The line where the issue ends.
  - end_column - The column where the issue ends.
  - severity - The severity of the issue.
  - message - The message of the result.
  - origin - The origin of the issue.
  - additional_info - Additional info provided by the issue.

  The groups `line`, `column`, `end_line` and `end_column` don't have to match numbers only, they can also match nothing, the generated `Result` is filled automatically with `None` then for the appropriate properties.

  Needs to be provided if `output_format` is `'regex'`.
- **severity_map** – A dict used to map a severity string (captured from the `output_regex` with the named group `severity`) to an actual

coalib.results.RESULT_SEVERITY for a result. Severity strings are mapped **case-insensitive**!

- RESULT_SEVERITY.MAJOR: Mapped by critical, c, fatal, fail, f,‘‘error‘‘, err or e.
- RESULT_SEVERITY.NORMAL: Mapped by warning, warn or w.
- RESULT_SEVERITY.INFO`: Mapped by ``information, info, i, note or suggestion.

A ValueError is raised when the named group severity is not used inside output_regex and this parameter is given.

- **diff_severity** – The severity to use for all results if output_format is 'corrected'. By default this value is coalib.results.RESULT_SEVERITY.NORMAL. The given value needs to be defined inside coalib.results.RESULT_SEVERITY.
- **result_message** – The message-string to use for all results. Can be used only together with corrected or regex output format. When using corrected, the default value is "Inconsistency found.", while for regex this static message is disabled and the message matched by output_regex is used instead.
- **diff_distance** – Number of unchanged lines that are allowed in between two changed lines so they get yielded as one diff if corrected output-format is given. If a negative distance is given, every change will be yielded as an own diff, even if they are right beneath each other. By default this value is 1.

Raises

- **ValueError** – Raised when invalid options are supplied.
- **TypeError** – Raised when incompatible types are supplied. See parameter documentations for allowed types.

Returns A LocalBear derivation that lints code using an external tool.

coalib.bearlib.abstractions.SectionCreatable module

class coalib.bearlib.abstractions.SectionCreatable.**SectionCreatable**

Bases: object

A SectionCreatable is an object that is creatable out of a section object. Thus this is the class for many helper objects provided by the bearlib.

If you want to use an object that inherits from this class the following approach is recommended: Instantiate it via the from_section method. You can provide default arguments via the lower case keyword arguments.

Example:

```
SpacingHelper.from_section(section, tabwidth=8)
```

creates a SpacingHelper and if the “tabwidth” setting is needed and not contained in section, 8 will be taken.

It is recommended to write the prototype of the __init__ method according to this example:

```
def __init__(self, setting_one: int, setting_two: bool=False):
    pass  # Implementation
```

This way the get_optional_settings and the get_non_optional_settings method will extract automatically that:

- setting_one should be an integer
- setting_two should be a bool and defaults to False

If you write a documentation comment, you can use :param to add descriptions to your parameters. These will be available too automatically.

classmethod **from_section**(*section*, ***kwargs*)
Creates the object from a section object.

Parameters

- **section** – A section object containing at least the settings specified by get_non_optional_settings()
- **kwargs** – Additional keyword arguments

classmethod **get_metadata**()

classmethod **get_non_optional_settings**()
Retrieves the minimal set of settings that need to be defined in order to use this object.

Returns a dictionary of needed settings as keys and help texts as values

classmethod **get_optional_settings**()
Retrieves the settings needed IN ADDITION to the ones of get_non_optional_settings to use this object without internal defaults.

Returns a dictionary of needed settings as keys and help texts as values

Module contents

The abstractions package contains classes that serve as interfaces for helper classes in the bearlib.

coalib.bearlib.aspects package

Submodules

coalib.bearlib.aspects.Metadata module

class coalib.bearlib.aspects.Metadata.**Body**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Body*, *coalib.bearlib.aspects.base.aspectbase*

class **Existence**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Existence*, *coalib.bearlib.aspects.base.aspectba*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Body*

subaspects = {}

class Body.**Length**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Length*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Body*

subaspects = {}

Body.**docs** = <coalib.bearlib.aspects.docs.Documentation object>

Body.**parent**
alias of *CommitMessage*

Body.**subaspects** = {'Existence': <aspectclass 'Root.Metadata.CommitMessage.Body.Existence'>, 'Length': <aspectcl

class coalib.bearlib.aspects.Metadata.**ColonExistence**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.ColonExistence*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class coalib.bearlib.aspects.Metadata.**CommitMessage**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.CommitMessage*, *coalib.bearlib.aspects.base.aspectbase*

class Body(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Body*, *coalib.bearlib.aspects.base.aspectbase*

class Existence(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Existence*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Body*

subaspects = {}

class CommitMessage.Body.**Length**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Length*, *coalib.bearlib.aspects.base.aspectba*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Body*

subaspects = {}

CommitMessage.Body.**docs** = <coalib.bearlib.aspects.docs.Documentation object>

CommitMessage.Body.**parent**
alias of *CommitMessage*

CommitMessage.Body.**subaspects** = {'Existence': <aspectclass 'Root.Metadata.CommitMessage.Body.Existen

class CommitMessage.**Emptiness**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Emptiness*, *coalib.bearlib.aspects.base.aspectba*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *CommitMessage*

subaspects = {}

class CommitMessage.**Shortlog**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Shortlog*, *coalib.bearlib.aspects.base.aspectbas*

class ColonExistence(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.ColonExistence*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class CommitMessage.Shortlog.**FirstCharacter**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.FirstCharacter*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class CommitMessage.Shortlog.**Length**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Length*, *coalib.bearlib.aspects.base.aspectba*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class CommitMessage.Shortlog.**Tense**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Tense*, *coalib.bearlib.aspects.base.aspectbas*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class CommitMessage.Shortlog.**TrailingPeriod**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.TrailingPeriod*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

CommitMessage.Shortlog.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

CommitMessage.Shortlog.**parent**
alias of *CommitMessage*

CommitMessage.Shortlog.**subaspects** = {'FirstCharacter': <aspectclass 'Root.Metadata.CommitMessage.Sh

CommitMessage.**docs** = <coalib.bearlib.aspects.docs.Documentation object>

CommitMessage.**parent**
alias of *Metadata*

CommitMessage.**subaspects** = {'Body': <aspectclass 'Root.Metadata.CommitMessage.Body'>, 'Emptiness': <aspect

class coalib.bearlib.aspects.Metadata.**Emptiness**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Emptiness*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *CommitMessage*

subaspects = {}

class coalib.bearlib.aspects.Metadata.**Existence**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Existence*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Body*

subaspects = {}

class coalib.bearlib.aspects.Metadata.**FirstCharacter**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.FirstCharacter*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class coalib.bearlib.aspects.Metadata.**Length**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Length*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Body*

subaspects = {}

class coalib.bearlib.aspects.Metadata.**Metadata**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Metadata*, *coalib.bearlib.aspects.base.aspectbase*

class CommitMessage(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.CommitMessage*, *coalib.bearlib.aspects.base.aspectbase*

class Body (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Body*, *coalib.bearlib.aspects.base.aspectbase*

class Existence (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Existence*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Body*

subaspects = {}

class Metadata.CommitMessage.Body.**Length** (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Length*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Body*

subaspects = {}

Metadata.CommitMessage.Body.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

Metadata.CommitMessage.Body.**parent**
alias of *CommitMessage*

Metadata.CommitMessage.Body.**subaspects = {'Existence': <aspectclass 'Root.Metadata.CommitMess**

class Metadata.CommitMessage.**Emptiness** (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Emptiness*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *CommitMessage*

subaspects = {}

class Metadata.CommitMessage.**Shortlog** (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Shortlog*, *coalib.bearlib.aspects.base.aspectbase*

class ColonExistence (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.ColonExistence*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class Metadata.CommitMessage.Shortlog.**FirstCharacter** (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.FirstCharacter*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class Metadata.CommitMessage.Shortlog.**Length**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Length*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class Metadata.CommitMessage.Shortlog.**Tense**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Tense*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class Metadata.CommitMessage.Shortlog.**TrailingPeriod**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.TrailingPeriod*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

Metadata.CommitMessage.Shortlog.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

Metadata.CommitMessage.Shortlog.**parent**
alias of *CommitMessage*

Metadata.CommitMessage.Shortlog.**subaspects = {'FirstCharacter': <aspectclass 'Root.Metadata.C**

Metadata.CommitMessage.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

Metadata.CommitMessage.**parent**
alias of *Metadata*

Metadata.CommitMessage.**subaspects = {'Body': <aspectclass 'Root.Metadata.CommitMessage.Body'>, 'E**

Metadata.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

Metadata.**parent**
alias of Root

Metadata.**subaspects = {'CommitMessage': <aspectclass 'Root.Metadata.CommitMessage'>}**

class coalib.bearlib.aspects.Metadata.**Shortlog**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Shortlog*, *coalib.bearlib.aspects.base.aspectbase*

class **ColonExistence**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.ColonExistence*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class Shortlog.**FirstCharacter**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.FirstCharacter*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class Shortlog.**Length**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Length*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class Shortlog.**Tense**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Tense*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class Shortlog.**TrailingPeriod**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.TrailingPeriod*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

Shortlog.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

Shortlog.**parent**
alias of *CommitMessage*

Shortlog.**subaspects = {'FirstCharacter': <aspectclass 'Root.Metadata.CommitMessage.Shortlog.FirstCharacter'>**

class coalib.bearlib.aspects.Metadata.**Tense**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Tense*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

class coalib.bearlib.aspects.Metadata.**TrailingPeriod**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.TrailingPeriod*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Shortlog*

subaspects = {}

coalib.bearlib.aspects.Redundancy module

class coalib.bearlib.aspects.Redundancy.**Clone**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.Clone*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Redundancy*

subaspects = {}

class coalib.bearlib.aspects.Redundancy.**Redundancy**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.Redundancy*, *coalib.bearlib.aspects.base.aspectbas*

class Clone(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.Clone*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Redundancy*

subaspects = {}

class Redundancy.**UnreachableCode**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnreachableCode*, *coalib.bearlib.aspects.base.aspectbase*

class UnreachableStatement(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnreachableStatement*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnreachableCode*

subaspects = {}

class Redundancy.UnreachableCode.**UnusedFunction**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedFunction*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnreachableCode*

subaspects = {}

Redundancy.UnreachableCode.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

Redundancy.UnreachableCode.**parent**
alias of *Redundancy*

Redundancy.UnreachableCode.**subaspects = {'UnreachableStatement': <aspectclass 'Root.Redundancy.Un**

class Redundancy.**UnusedImport**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedImport*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Redundancy*

subaspects = {}

class Redundancy.**UnusedVariable**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedVariable*, *coalib.bearlib.aspects.base.aspectbase*

class UnusedGlobalVariable(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedGlobalVariable*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnusedVariable*

subaspects = {}

class Redundancy.UnusedVariable.**UnusedLocalVariable**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedLocalVariable*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnusedVariable*

subaspects = {}

class Redundancy.UnusedVariable.**UnusedParameter**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedParameter*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnusedVariable*

subaspects = {}

Redundancy.UnusedVariable.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

Redundancy.UnusedVariable.**parent**
alias of *Redundancy*

Redundancy.UnusedVariable.**subaspects = {'UnusedLocalVariable': <aspectclass 'Root.Redundancy.Unus**

Redundancy.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

Redundancy.**parent**
alias of Root

Redundancy.**subaspects = {'Clone': <aspectclass 'Root.Redundancy.Clone'>, 'UnreachableCode': <aspectclass 'Roo**

class coalib.bearlib.aspects.Redundancy.**UnreachableCode**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnreachableCode*, *coalib.bearlib.aspects.base.aspectbase*

class UnreachableStatement(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnreachableStatement*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnreachableCode*

subaspects = {}

class UnreachableCode.**UnusedFunction**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedFunction*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnreachableCode*

subaspects = {}

UnreachableCode.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

UnreachableCode.**parent**
alias of *Redundancy*

UnreachableCode.**subaspects = {'UnreachableStatement': <aspectclass 'Root.Redundancy.UnreachableCode.Unre**

class coalib.bearlib.aspects.Redundancy.**UnreachableStatement**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnreachableStatement*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnreachableCode*

subaspects = {}

class coalib.bearlib.aspects.Redundancy.**UnusedFunction**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedFunction*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnreachableCode*

subaspects = {}

class coalib.bearlib.aspects.Redundancy.**UnusedGlobalVariable**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedGlobalVariable*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnusedVariable*

subaspects = {}

class coalib.bearlib.aspects.Redundancy.**UnusedImport**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedImport*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Redundancy*

subaspects = {}

class coalib.bearlib.aspects.Redundancy.**UnusedLocalVariable**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedLocalVariable*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnusedVariable*

subaspects = {}

class coalib.bearlib.aspects.Redundancy.**UnusedParameter**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedParameter*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnusedVariable*

subaspects = {}

class coalib.bearlib.aspects.Redundancy.**UnusedVariable**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedVariable*, *coalib.bearlib.aspects.base.aspectbase*

class UnusedGlobalVariable(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedGlobalVariable*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnusedVariable*

subaspects = {}

class UnusedVariable.**UnusedLocalVariable**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedLocalVariable*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnusedVariable*

subaspects = {}

class UnusedVariable.**UnusedParameter**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedParameter*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *UnusedVariable*

subaspects = {}

UnusedVariable.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

UnusedVariable.**parent**
alias of *Redundancy*

UnusedVariable.**subaspects = {'UnusedLocalVariable': <aspectclass 'Root.Redundancy.UnusedVariable.UnusedL**

coalib.bearlib.aspects.base module

class coalib.bearlib.aspects.base.**aspectbase**(*language*, ***taste_values*)
Bases: object

Base class for aspectclasses with common features for their instances.

Derived classes must use coalib.bearlib.aspectclasses.meta.aspectclass as metaclass. This is automatically handled by coalib.bearlib.aspectclasses.meta.aspectclass.subaspect() decorator.

tastes
Get a dictionary of all taste names mapped to their specific values, including parent tastes.

coalib.bearlib.aspects.docs module

class coalib.bearlib.aspects.docs.**Documentation**(*definition: str = ''*, *example: str = ''*, *example_language: str = ''*, *importance_reason: str = ''*, *fix_suggestions: str = ''*)
Bases: object

This class contains documentation about an aspectclass. The documentation is consistent if all members are given:

```
>>> Documentation('defined').check_consistency()
False
>>> Documentation('definition', 'example',
...               'example_language', 'importance',
...               'fix').check_consistency()
True
```

check_consistency()

coalib.bearlib.aspects.meta module

class coalib.bearlib.aspects.meta.**aspectclass**(*clsname*, *bases*, *clsattrs*)
Bases: type

Metaclass for aspectclasses.

Root aspectclass is coalib.bearlib.aspectclasses.Root.

subaspect(*subcls*)
The sub-aspectclass decorator.

See coalib.bearlib.aspectclasses.Root for description and usage.

tastes
Get a dictionary of all taste names mapped to their coalib.bearlib.aspectclasses.Taste instances.

coalib.bearlib.aspects.taste module

coalib.bearlib.aspects.taste.**Taste**
Defines tastes in aspectclass definitions.

Tastes can be made only available for certain languages by providing a tuple of language identifiers on instantiation:

```
>>> Taste[bool](
...     'Ignore ``using`` directives in C#.',
...     (True, False), default=False,
...     languages=('CSharp', )
... ).languages
(C#,)
```

If no *languages* are given, they will be available for any language. See coalib.bearlib.aspectclasses.Root for further usage.

exception coalib.bearlib.aspects.taste.**TasteError**
Bases: AttributeError

A taste is not allowed to be accessed.

class coalib.bearlib.aspects.taste.**TasteMeta**
Bases: type

Metaclass for coalib.bearlib.aspectclasses.Taste

Allows defining taste cast type via __getitem__(), like:

```
>>> Taste[int]().cast_type
<class 'int'>
```

Module contents

class coalib.bearlib.aspects.**Root**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.base.aspectbase*

The root aspectclass.

Define sub-aspectclasses with class-bound `.subaspect` decorator. Definition string is taken from doc-string of decorated class. Remaining docs are taken from a nested `docs` class. Tastes are defined as class attributes that are instances of `coalib.bearlib.aspectclasses.Taste`.

```
>>> @Root.subaspect
... class Formatting:
...     """
...     A parent aspect for code formatting aspects...
...     """
```

We can now create subaspects like this:

```
>>> @Formatting.subaspect
... class LineLength:
...     """
...     This aspect controls the length of a line...
...     """
...     class docs:
...        example = "..."
...        example_language = "..."
...        importance_reason = "..."
...        fix_suggestions = "..."
...
...     max_line_length = Taste[int](
...         "Maximum length allowed for a line.",
...         (80, 90, 120), default=80)
```

The representation will show the full "path" to the leaf of the tree:

```
>>> Root.Formatting.LineLength
<aspectclass 'Root.Formatting.LineLength'>
```

We can see, which settings are availables:

```
>>> Formatting.tastes
{}
>>> LineLength.tastes
{'max_line_length': <....Taste[int] object at ...>}
```

And instantiate the aspect with the values, they will be automatically converted:

```
>>> Formatting('Python')
<coalib.bearlib.aspects.Root.Formatting object at 0x...>
>>> LineLength('Python', max_line_length="100").tastes
{'max_line_length': 100}
```

If no settings are given, the defaults will be taken> >>> LineLength('Python').tastes {'max_line_length': 80}

Tastes can also be made available for only specific languages:

```
>>> from coalib.bearlib.languages import Language
>>> @Language
... class GreaterTrumpScript:
...     pass
```

```
>>> @Formatting.subaspect
... class Greatness:
...     """
...     This aspect controls the greatness of a file...
...     """
...
...     min_greatness = Taste[int](
...         "Minimum greatness factor needed for a TrumpScript file. "
...         "This is fact.",
...         (1000000, 1000000000, 1000000000000), default=1000000,
...         languages=('GreaterTrumpScript' ,))
```

```
>>> Greatness.tastes
{'min_greatness': <....Taste[int] object at ...>}
>>> Greatness('GreaterTrumpScript').tastes
{'min_greatness': 1000000}
>>> Greatness('GreaterTrumpScript', min_greatness=1000000000000).tastes
{'min_greatness': 1000000000000}
```

```
>>> Greatness('Python').tastes
{}
```

```
>>> Greatness('Python', min_greatness=1000000000)
...
Traceback (most recent call last):
  ...
coalib.bearlib.aspects.taste.TasteError:
Root.Formatting.Greatness.min_greatness is not available ...
```

```
>>> Greatness('Python').min_greatness
...
Traceback (most recent call last):
  ...
coalib.bearlib.aspects.taste.TasteError:
Root.Formatting.Greatness.min_greatness is not available ...
```

class Metadata (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Metadata*, *coalib.bearlib.aspects.base.aspectbas*

class CommitMessage (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.CommitMessage*, *coalib.bearlib.aspects.base.aspectbase*

class Body (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Body*, *coalib.bearlib.aspects.base.aspectbase*

class Existence (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Existence*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of `Body`

subaspects = {}

class Root.Metadata.CommitMessage.Body.**Length**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Length*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of Body

subaspects = {}

Root.Metadata.CommitMessage.Body.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

Root.Metadata.CommitMessage.Body.**parent**
alias of CommitMessage

Root.Metadata.CommitMessage.Body.**subaspects = {'Existence': <aspectclass 'Root.Metadata.C**

class Root.Metadata.CommitMessage.**Emptiness**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Emptiness*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of CommitMessage

subaspects = {}

class Root.Metadata.CommitMessage.**Shortlog**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Shortlog*, *coalib.bearlib.aspects.base.aspectbase*

class ColonExistence(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.ColonExistence*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of Shortlog

subaspects = {}

class Root.Metadata.CommitMessage.Shortlog.**FirstCharacter**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.FirstCharacter*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of Shortlog

subaspects = {}

class Root.Metadata.CommitMessage.Shortlog.**Length**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Length*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of Shortlog

subaspects = {}

class Root.Metadata.CommitMessage.Shortlog.**Tense**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.Tense*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of Shortlog

subaspects = {}

class Root.Metadata.CommitMessage.Shortlog.**TrailingPeriod**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Metadata.TrailingPeriod*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of Shortlog

subaspects = {}

Root.Metadata.CommitMessage.Shortlog.**docs** = <coalib.bearlib.aspects.docs.Documentation obj

Root.Metadata.CommitMessage.Shortlog.**parent**
alias of CommitMessage

Root.Metadata.CommitMessage.Shortlog.**subaspects** = {'FirstCharacter': <aspectclass 'Root.M

Root.Metadata.CommitMessage.**docs** = <coalib.bearlib.aspects.docs.Documentation object>

Root.Metadata.CommitMessage.**parent**
alias of *Metadata*

Root.Metadata.CommitMessage.**subaspects** = {'Body': <aspectclass 'Root.Metadata.CommitMessage.

Root.Metadata.**docs** = <coalib.bearlib.aspects.docs.Documentation object>

Root.Metadata.**parent**
alias of *Root*

Root.Metadata.**subaspects** = {'CommitMessage': <aspectclass 'Root.Metadata.CommitMessage'>}

class Root.**Redundancy**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.Redundancy*, *coalib.bearlib.aspects.base.aspectbase*

class **Clone**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.Clone*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Redundancy*

subaspects = {}

class Root.Redundancy.**UnreachableCode**(*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnreachableCode*, *coalib.bearlib.aspects.base.aspectbase*

class UnreachableStatement (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnreachableStatement*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of UnreachableCode

subaspects = {}

class Root.Redundancy.UnreachableCode.**UnusedFunction** (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedFunction*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of UnreachableCode

subaspects = {}

Root.Redundancy.UnreachableCode.**docs = <coalib.bearlib.aspects.docs.Documentation object>**

Root.Redundancy.UnreachableCode.**parent**
alias of *Redundancy*

Root.Redundancy.UnreachableCode.**subaspects = {'UnreachableStatement': <aspectclass 'Root.Red**

class Root.Redundancy.**UnusedImport** (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedImport*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of *Redundancy*

subaspects = {}

class Root.Redundancy.**UnusedVariable** (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedVariable*, *coalib.bearlib.aspects.base.aspectbase*

class UnusedGlobalVariable (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedGlobalVariable*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of UnusedVariable

subaspects = {}

class Root.Redundancy.UnusedVariable.**UnusedLocalVariable** (*language*, ***taste_values*)
Bases: *coalib.bearlib.aspects.Redundancy.UnusedLocalVariable*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of UnusedVariable

subaspects = {}

class Root.Redundancy.UnusedVariable.**UnusedParameter**(*language*, ***taste_values*)

Bases: *coalib.bearlib.aspects.Redundancy.UnusedParameter*, *coalib.bearlib.aspects.base.aspectbase*

docs = <coalib.bearlib.aspects.docs.Documentation object>

parent
alias of UnusedVariable

subaspects = {}

Root.Redundancy.UnusedVariable.**docs** = <coalib.bearlib.aspects.docs.Documentation object>

Root.Redundancy.UnusedVariable.**parent**
alias of *Redundancy*

Root.Redundancy.UnusedVariable.**subaspects** = {'UnusedLocalVariable': <aspectclass 'Root.Redun

Root.Redundancy.**docs** = <coalib.bearlib.aspects.docs.Documentation object>

Root.Redundancy.**parent**
alias of *Root*

Root.Redundancy.**subaspects** = {'Clone': <aspectclass 'Root.Redundancy.Clone'>, 'UnreachableCode': <asp

Root.**parent** = None

Root.**subaspects** = {'Metadata': <aspectclass 'Root.Metadata'>, 'Redundancy': <aspectclass 'Root.Redundancy'>}

coalib.bearlib.aspects.**Taste**
Defines tastes in aspectclass definitions.

Tastes can be made only available for certain languages by providing a tuple of language identifiers on instantiation:

```
>>> Taste[bool](
...     'Ignore ``using`` directives in C#.',
...     (True, False), default=False,
...     languages=('CSharp', )
... ).languages
(C#,)
```

If no *languages* are given, they will be available for any language. See coalib.bearlib.aspectclasses.Root for further usage.

exception coalib.bearlib.aspects.**TasteError**
Bases: AttributeError

A taste is not allowed to be accessed.

class coalib.bearlib.aspects.**aspectclass**(*clsname*, *bases*, *clsattrs*)
Bases: type

Metaclass for aspectclasses.

Root aspectclass is coalib.bearlib.aspectclasses.Root.

subaspect(*subcls*)
The sub-aspectclass decorator.

See coalib.bearlib.aspectclasses.Root for description and usage.

tastes
Get a dictionary of all taste names mapped to their `coalib.bearlib.aspectclasses.Taste` instances.

coalib.bearlib.languages package

Subpackages

coalib.bearlib.languages.definitions package

Submodules

coalib.bearlib.languages.definitions.C module

coalib.bearlib.languages.definitions.CPP module

coalib.bearlib.languages.definitions.CSS module

coalib.bearlib.languages.definitions.CSharp module

coalib.bearlib.languages.definitions.Java module

coalib.bearlib.languages.definitions.JavaScript module

coalib.bearlib.languages.definitions.Python module

coalib.bearlib.languages.definitions.Vala module

Module contents

This directory holds language definitions.

Language definitions hold expressions that help defining specific syntax elements for a programming language.

Currently defined keys are:

names extensions comment_delimiter multiline_comment_delimiters string_delimiters multiline_string_delimiters keywords special_chars

coalib.bearlib.languages.documentation package

Submodules

coalib.bearlib.languages.documentation.DocstyleDefinition module

class coalib.bearlib.languages.documentation.DocstyleDefinition.**DocstyleDefinition**(*language: str*, *docstyle: str*, *markers: (<class 'collections.abc.Itera*, *<class 'str'>)*, *metadata: coalib.bearlib.*

Bases: object

The DocstyleDefinition class holds values that identify a certain type of documentation comment (for which language, documentation style/tool used etc.).

class **Metadata**(*param_start*, *param_end*, *return_sep*)

Bases: tuple

param_end
Alias for field number 1

param_start
Alias for field number 0

return_sep
Alias for field number 2

DocstyleDefinition.**docstyle**
The documentation style/tool used to document code.

Returns A lower-case string defining the docstyle (i.e. "default" or "doxygen").

static DocstyleDefinition.**get_available_definitions**()
Returns a sequence of pairs with (docstyle, language) which are available when using load().

Returns A sequence of pairs with (docstyle, language).

DocstyleDefinition.**language**
The programming language.

Returns A lower-case string defining the programming language (i.e. "cpp" or "python").

classmethod DocstyleDefinition.**load**(*language: str*, *docstyle: str*, *coalang_dir=None*)
Loads a DocstyleDefinition from the coala docstyle definition files.

This function considers all settings inside the according coalang-files as markers, except param_start, param_end and return_sep which are considered as special metadata markers.

Note: When placing new coala docstyle definition files, these must consist of only lowercase letters and end with `.coalang`!

Parameters

- **language** – The case insensitive programming language of the documentation comment as a string.
- **docstyle** – The case insensitive documentation style/tool used to document code, e.g. `"default"` or `"doxygen"`.
- **coalang_dir** – Path to directory with coalang docstyle definition files. This replaces the default path if given.

Raises

- **FileNotFoundError** – Raised when the given docstyle was not found.
- **KeyError** – Raised when the given language is not defined for given docstyle.

Returns The `DocstyleDefinition` for given language and docstyle.

`DocstyleDefinition.`**markers**

A tuple of marker sets that identify a documentation comment.

Marker sets consist of 3 entries where the first is the start-marker, the second one the each-line marker and the last one the end-marker. For example a marker tuple with a single marker set `(("/**","*","*/"),)` would match following documentation comment:

```
/**
 * This is documentation.
 */
```

It's also possible to supply an empty each-line marker (`("/**","","*/")`):

```
/**
 This is more documentation.
 */
```

Markers are matched "greedy", that means it will match as many each-line markers as possible. I.e. for `("///","///","///")`):

```
/// Brief documentation.
///
/// Detailed documentation.
```

Returns A tuple of marker/delimiter string tuples that identify a documentation comment.

`DocstyleDefinition.`**metadata**

A namedtuple of certain attributes present in the documentation.

These attributes are used to define parts of the documentation.

coalib.bearlib.languages.documentation.DocumentationComment module

class coalib.bearlib.languages.documentation.DocumentationComment.**DocumentationComment**(*documen*
doc-
style_de
in-
dent,
marker,
range)

Bases: object

The DocumentationComment holds information about a documentation comment inside source-code, like position etc.

class **Description**(*desc*)

Bases: tuple

desc

Alias for field number 0

class DocumentationComment.**Parameter**(*name*, *desc*)

Bases: tuple

desc

Alias for field number 1

name

Alias for field number 0

class DocumentationComment.**ReturnValue**(*desc*)

Bases: tuple

desc

Alias for field number 0

DocumentationComment.**assemble**()

Assembles parsed documentation to the original documentation.

This function assembles the whole documentation comment, with the given markers and indentation.

DocumentationComment.**docstyle**

classmethod DocumentationComment.**from_metadata**(*doccomment*, *docstyle_definition*, *marker*, *indent*, *range*)

Assembles a list of parsed documentation comment metadata.

This function just assembles the documentation comment itself, without the markers and indentation.

```
>>> from coalib.bearlib.languages.documentation.DocumentationComment \
...     import DocumentationComment
>>> from coalib.bearlib.languages.documentation.DocstyleDefinition \
...     import DocstyleDefinition
>>> from coalib.results.TextRange import TextRange
>>> Description = DocumentationComment.Description
>>> Parameter = DocumentationComment.Parameter
>>> python_default = DocstyleDefinition.load("python3", "default")
>>> parsed_doc = [Description(desc='\nDescription\n'),
...               Parameter(name='age', desc=' Age\n')]
>>> str(DocumentationComment.from_metadata(
...         parsed_doc, python_default,
...         python_default.markers[0], 4,
```

```
...         TextRange.from_values(0, 0, 0, 0)))
'\nDescription\n:param age: Age\n'
```

Parameters

- **doccomment** – The list of parsed documentation comment metadata.
- **docstyle_definition** – The DocstyleDefinition instance that defines what docstyle is being used in a documentation comment.
- **marker** – The markers to be used in the documentation comment.
- **indent** – The indentation to be used in the documentation comment.
- **range** – The range of the documentation comment.

Returns A DocumentationComment instance of the assembled documentation.

DocumentationComment.**language**

DocumentationComment.**metadata**

DocumentationComment.**parse**()

Parses documentation independent of language and docstyle.

Returns The list of all the parsed sections of the documentation. Every section is a namedtuple of either Description or Parameter or ReturnValue.

Raises **NotImplementedError** – When no parsing method is present for the given language and docstyle.

coalib.bearlib.languages.documentation.DocumentationExtraction module

coalib.bearlib.languages.documentation.DocumentationExtraction.**extract_documentation**(*content*, *language*, *docstyle*)

Extracts all documentation texts inside the given source-code-string using the coala docstyle definition files.

The documentation texts are sorted by their order appearing in content.

For more information about how documentation comments are identified and extracted, see DocstyleDefinition.doctypes enumeration.

Parameters

- **content** – The source-code-string where to extract documentation from. Needs to be a list or tuple where each string item is a single line (including ending whitespaces like \n).
- **language** – The programming language used.
- **docstyle** – The documentation style/tool used (e.g. doxygen).

Raises

- **FileNotFoundError** – Raised when the docstyle definition file was not found.
- **KeyError** – Raised when the given language is not defined in given docstyle.
- **ValueError** – Raised when a docstyle definition setting has an invalid format.

Returns An iterator returning each DocumentationComment found in the content.

coalib.bearlib.languages.documentation.DocumentationExtraction.**extract_documentation_with_r**

Extracts all documentation texts inside the given source-code-string.

Parameters

- **content** – The source-code-string where to extract documentation from. Needs to be a list or tuple where each string item is a single line (including ending whitespaces like \n).
- **markers** – The list/tuple of marker-sets that identify a documentation-comment. Low-index markers have higher priority than high-index markers.

Returns An iterator returning each DocumentationComment found in the content.

Module contents

Provides facilities to extract, parse and assemble documentation comments for different languages and documentation tools.

Submodules

coalib.bearlib.languages.Language module

class coalib.bearlib.languages.Language.**Language**(**versions*)

Bases: object

This class defines programming languages and their versions.

You can define a new programming language as follows:

```
>>> @Language
... class TrumpScript:
...     __qualname__ = "America is great."
...     aliases = 'ts',
...     versions = 2.7, 3.3, 3.4, 3.5, 3.6
...     comment_delimiter = '#'
...     string_delimiter = {"'": "'"}
```

From a bear, you can simply parse the user given language string to get the instance of the Language you desire:

```
>>> Language['trumpscript']
America is great. 2.7, 3.3, 3.4, 3.5, 3.6
>>> Language['ts 3.4, 3.6']
America is great. 3.4, 3.6
>>> Language['TS 3']
America is great. 3.3, 3.4, 3.5, 3.6
>>> Language['tS 1']
Traceback (most recent call last):
 ...
ValueError: No versions left
```

The attributes are not accessible unless you have selected one - and only one - version of your language:

```
>>> Language.TrumpScript(3.3, 3.4).comment_delimiter
Traceback (most recent call last):
 ...
```

```
AttributeError: You have to specify ONE version ...
>>> Language.TrumpScript(3.3).comment_delimiter
'#'
```

If you don't know which version is the right one, just use this:

```
>>> Language.TrumpScript().get_default_version()
America is great. 3.6
```

To see which attributes are available, use the `attributes` property:

```
>>> Language.TrumpScript(3.3).attributes
['comment_delimiter', 'string_delimiter']
```

You can access a dictionary of the attribute values for every version from the class:

```
>>> Language.TrumpScript.comment_delimiter
OrderedDict([(2.7, '#'), (3.3, '#'), (3.4, '#'), (3.5, '#'), (3.6, '#')])
```

Any nonexistent item will of course not be served:

```
>>> Language.TrumpScript.unknown_delimiter
Traceback (most recent call last):
  ...
AttributeError
```

You now know the most important parts for writing a bear using languages. Read ahead if you want to know more about working with multiple versions of programming languages as well as derivative languages!

We can define derivative languages as follows:

```
>>> @Language
... class TrumpScriptDerivative(Language.TrumpScript):
...     __qualname__ = 'Shorter'
...     comment_delimiter = '//'
...     keywords = None
```

```
>>> Language.TrumpScriptDerivative()
Shorter 2.7, 3.3, 3.4, 3.5, 3.6
```

```
>>> Language.TrumpScriptDerivative().get_default_version().attributes
['comment_delimiter', 'keywords', 'string_delimiter']
>>> Language.TrumpScriptDerivative().get_default_version().keywords
>>> Language.TrumpScriptDerivative().get_default_version().comment_delimiter
'//'
>>> Language.TrumpScriptDerivative().get_default_version().string_delimiter
{"'": "'"}
```

We can get an instance via this syntax as well:

```
>>> Language[Language.TrumpScript]
America is great. 2.7, 3.3, 3.4, 3.5, 3.6
>>> Language[Language.TrumpScript(3.6)]
America is great. 3.6
```

As you see, you can use the *__qualname__* property. This will also affect the string representation and work as an implicit alias:

```
>>> str(Language.TrumpScript(3.4))
'America is great. 3.4'
```

We can specify the version by instantiating the TrumpScript class now:

```
>>> str(Language.TrumpScript(3.6))
'America is great. 3.6'
```

You can also define ranges of versions of languages:

```
>>> (Language.TrumpScript > 3.3) <= 3.5
America is great. 3.4, 3.5
```

```
>>> Language.TrumpScript == 3
America is great. 3.3, 3.4, 3.5, 3.6
```

Those can be combined by the or operator:

```
>>> (Language.TrumpScript == 3.6) | (Language.TrumpScript == 2)
America is great. 2.7, 3.6
```

The *__contains__* operator of the class is defined as well for strings and instances. This is case insensitive and aliases are allowed:

```
>>> Language.TrumpScript(3.6) in Language.TrumpScript
True
>>> 'ts 3.6, 3.5' in Language.TrumpScript
True
>>> 'TrumpScript 2.6' in Language.TrumpScript
False
>>> 'TrumpScript' in Language.TrumpScript
True
```

This also works on instances:

```
>>> 'ts 3.6, 3.5' in (Language.TrumpScript == 3)
True
>>> 'ts 3.6,3.5' in ((Language.TrumpScript == 2)
...                  | Language.TrumpScript(3.5))
False
>>> Language.TrumpScript(2.7, 3.5) in (Language.TrumpScript == 3)
False
>>> Language.TrumpScript(3.5) in (Language.TrumpScript == 3)
True
```

Any undefined language will obviously not be available:

```
>>> Language.Cobol
Traceback (most recent call last):
 ...
AttributeError
```

attributes
Retrieves the names of all attributes that are available for this language.

get_default_version()
Retrieves the latest version the user would want to choose from the given versions in self.

(At a later point this might also retrieve a default version specifiable by the language definition, so keep using this!)

class coalib.bearlib.languages.Language.**LanguageMeta**
Bases: type

Metaclass for *coalib.bearlib.languages.Language.Language*.

Allows it being used as a decorator as well as implements the *__contains__* operation and stores all languages created with the decorator.

The operators are defined on the class as well, so you can do the following:

```
>>> @Language
... class SomeLang:
...     versions = 2.7, 3.3, 3.4, 3.5, 3.6
>>> Language.SomeLang > 3.4
SomeLang 3.5, 3.6
>>> Language.SomeLang < 3.4
SomeLang 2.7, 3.3
>>> Language.SomeLang >= 3.4
SomeLang 3.4, 3.5, 3.6
>>> Language.SomeLang <= 3.4
SomeLang 2.7, 3.3, 3.4
>>> Language.SomeLang == 3.4
SomeLang 3.4
>>> Language.SomeLang != 3.4
SomeLang 2.7, 3.3, 3.5, 3.6
>>> Language.SomeLang == 1.0
Traceback (most recent call last):
 ...
ValueError: No versions left
```

class coalib.bearlib.languages.Language.**LanguageUberMeta**
Bases: type

This class is used to hide the *all* attribute from the Language class.

all = [<class 'coalib.bearlib.languages.Language.C'>, <class 'coalib.bearlib.languages.Language.CPP'>, <class 'coalib.be

class coalib.bearlib.languages.Language.**Languages**
Bases: tuple

A tuple-based container for *coalib.bearlib.languages.Language* instances. It supports language identifiers in any format accepted by Language[...]:

```
>>> Languages(['C#', Language.Python == 3])
(C#, Python 3.3, 3.4, 3.5, 3.6)
```

It provides __contains__() for checking if a given language identifier is included:

```
>>> 'Python 2.7, 3.5' in Languages([Language.Python()])
True
>>> 'Py 3.3' in Languages(['Python 2'])
False
>>> 'csharp' in Languages(['C#', Language.Python == 3])
True
```

coalib.bearlib.languages.Language.**limit_versions**(*language*, *limit*, *operator*)
Limits given languages with the given operator:

Parameters

- **language** – A *Language* instance.
- **limit** – A number to limit the versions.
- **operator** – The operator to use for the limiting.

Returns A new *Language* instance with limited versions.

Raises **ValueError** – If no version is left anymore.

coalib.bearlib.languages.Language.**parse_lang_str**(*string*)
Prarses any given language string into name and a list of float versions:

```
>>> parse_lang_str("Python")
('Python', [])
>>> parse_lang_str("Python 3.3")
('Python', [3.3])
>>> parse_lang_str("Python 3.6, 3.3")
('Python', [3.6, 3.3])
>>> parse_lang_str("Objective C 3.6, 3.3")
('Objective C', [3.6, 3.3])
>>> parse_lang_str("Cobol, stupid!")  # +ELLIPSIS
Traceback (most recent call last):
 ...
ValueError: Couldn't convert value 'stupid!' ...
```

coalib.bearlib.languages.LanguageDefinition module

class coalib.bearlib.languages.LanguageDefinition.**LanguageDefinition**(*language: str*, *coalang_dir=None*)
Bases: *coalib.bearlib.abstractions.SectionCreatable.SectionCreatable*

This class is deprecated! Use the *Language* class instead.

A Language Definition holds constants which may help parsing the language. If you want to write a bear you'll probably want to use those definitions to keep your bear independent of the semantics of each language.

You can easily get your language definition by just creating it with the name of the language desired:

```
>>> list(LanguageDefinition("cpp")['extensions'])
['.c', '.cpp', '.h', '.hpp']
```

For some languages aliases exist, the name is case insensitive; they will behave just like before and return settings:

```
>>> dict(LanguageDefinition('C++')['comment_delimiter'])
{'//': ''}
>>> dict(LanguageDefinition('C++')['string_delimiters'])
{'"': '"'}
```

If no language exists, you will get a `FileNotFoundError`:

```
>>> LanguageDefinition("BULLSHIT!")
Traceback (most recent call last):
 ...
FileNotFoundError
```

Custom coalangs are no longer supported. You can simply register your languages to the Languages decorator. When giving a custom coalang directory a warning will be emitted and it will attempt to load the given Language anyway through conventional means:

```
>>> LanguageDefinition("custom", coalang_dir='somewhere')
Traceback (most recent call last):
 ...
FileNotFoundError
```

If you need a custom language, just go like this:

```
>>> @Language
... class MyLittlePony:
...     color = 'green'
...     legs = 5
>>> int(LanguageDefinition('mylittlepony')['legs'])
5
```

But seriously, just use *Language* - and mind that it's already typed:

```
>>> Language['mylittlepony'].get_default_version().legs
5
```

Module contents

This directory holds means to get generic information for specific languages.

coalib.bearlib.naming_conventions package

Module contents

coalib.bearlib.naming_conventions.**to_camelcase**(*string*)

Converts the given string to camel-case.

```
>>> to_camelcase('Hello_world')
'helloWorld'
>>> to_camelcase('__Init__file__')
'__initFile__'
>>> to_camelcase('')
''
>>> to_camelcase('alreadyCamelCase')
'alreadyCamelCase'
>>> to_camelcase('   string')
'___string'
```

Parameters **string** – The string to convert.

Returns The camel-cased string.

coalib.bearlib.naming_conventions.**to_pascalcase**(*string*)
Converts the given to string pascal-case.

```
>>> to_pascalcase('hello_world')
'HelloWorld'
>>> to_pascalcase('__init__file__')
'__InitFile__'
>>> to_pascalcase('')
''
>>> to_pascalcase('AlreadyPascalCase')
'AlreadyPascalCase'
>>> to_pascalcase('   string')
'___String'
```

Parameters **string** – The string to convert.

Returns The pascal-cased string.

coalib.bearlib.naming_conventions.**to_snakecase**(*string*)
Converts the given string to snake-case.

```
>>> to_snakecase('HelloWorld')
'hello_world'
>>> to_snakecase('__Init__File__')
'__init_file__'
>>> to_snakecase('')
''
>>> to_snakecase('already_snake_case')
'already_snake_case'
>>> to_snakecase('   string  ')
'___string__'
>>> to_snakecase('ABCde.F.G..H..IH')
'a_b_cde.f.g..h..i_h'
```

Parameters **string** – The string to convert.

Returns The snake-cased string.

coalib.bearlib.naming_conventions.**to_spacecase**(*string*)
Converts the given string to space-case.

```
>>> to_spacecase('helloWorld')
'Hello World'
>>> to_spacecase('__Init__File__')
'Init File'
>>> to_spacecase('')
''
>>> to_spacecase('Already Space Case')
'Already Space Case'
>>> to_spacecase('  string  ')
'String'
```

Parameters **string** – The string to convert.

Returns The space-cased string.

coalib.bearlib.spacing package

Submodules

coalib.bearlib.spacing.SpacingHelper module

class coalib.bearlib.spacing.SpacingHelper.**SpacingHelper**(*tab_width: int = 4*)

Bases: *coalib.bearlib.abstractions.SectionCreatable.SectionCreatable*

DEFAULT_TAB_WIDTH = 4

get_indentation(*line: str*)

Checks the lines indentation.

Parameters **line** – A string to check for indentation.

Returns The indentation count in spaces.

replace_spaces_with_tabs(*line: str*)

Replaces spaces with tabs where possible. However in no case only one space will be replaced by a tab.

Example: ” a_text another” will be converted to ” a_text another”, assuming the tab_width is set to 4.

Parameters **line** – The string with spaces to replace.

Returns The converted string.

replace_tabs_with_spaces(*line: str*)

Replaces tabs in this line with the appropriate number of spaces.

Example: ” ” will be converted to ” ”, assuming the tab_width is set to 4.

Parameters **line** – The string with tabs to replace.

Returns A string with no tabs.

yield_tab_lengths(*input: str*)

Yields position and size of tabs in a input string.

Parameters **input** – The string with tabs.

Module contents

Module contents

The bearlib is an optional library designed to ease the task of any Bear. Just as the rest of coala the bearlib is designed to be as easy to use as possible while offering the best possible flexibility.

coalib.bearlib.**deprecate_bear**(*bear*)

Use this to deprecate a bear. Say we have a bear:

```
>>> class SomeBear:
...     def run(*args):
...         print("I'm running!")
```

To change the name from `SomeOldBear` to `SomeBear` you can keep the `SomeOldBear.py` around with those contents:

```
>>> @deprecate_bear
... class SomeOldBear(SomeBear): pass
```

Now let's run the bear:

```
>>> import sys
>>> logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)
>>> SomeOldBear().run()
WARNING:root:The bear SomeOldBear is deprecated. Use SomeBear instead!
I'm running!
```

Parameters **bear** – An old bear class that inherits from the new one (so it gets its methods and can just contain a pass.)

Returns A bear class that warns about deprecation on use.

coalib.bearlib.**deprecate_settings**(***depr_args*)

The purpose of this decorator is to allow passing old settings names to bears due to the heavy changes in their names.

```
>>> @deprecate_settings(new='old')
... def run(new):
...     print(new)
```

Now we can simply call the bear with the deprecated setting, we'll get a warning - but it still works!

```
>>> import sys
>>> logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)
>>> run(old="Hello world!")
WARNING:root:The setting `old` is deprecated. Please use `new` instead.
Hello world!
>>> run(new="Hello world!")
Hello world!
```

This example represents the case where the old setting name needs to be modified to match the new one.

```
>>> @deprecate_settings(new=('old', lambda a: a + 'coala!'))
... def func(new):
...     print(new)
```

```
>>> func(old="Welcome to ")
WARNING:root:The setting `old` is deprecated. Please use `new` instead.
Welcome to coala!
>>> func(new='coala!')
coala!
```

This example represents the case where the old and new settings are provided to the function.

```
>>> @deprecate_settings(new='old')
... def run(new):
...     print(new)
>>>
... run(old="Hello!", new='coala is always written with lowercase `c`.')
WARNING:root:The setting `old` is deprecated. Please use `new` instead.
WARNING:root:The value of `old` and `new` are conflicting. `new` will...
coala is always written with lowercase `c`.
>>> run(old='Hello!', new='Hello!')
WARNING:root:The setting `old` is deprecated. Please use `new` instead.
Hello!
```

The metadata for coala has been adjusted as well:

```
>>> list(run.__metadata__.non_optional_params.keys())
['new']
>>> list(run.__metadata__.optional_params.keys())
['old']
```

You cannot deprecate an already deprecated setting. Don't try. It will introduce non-deterministic errors in your program.

Parameters **depr_args** – A dictionary of settings as keys and their deprecated names as values.

1.1.2 coalib.bears package

Subpackages

coalib.bears.requirements package

Submodules

coalib.bears.requirements.DistributionRequirement module

class coalib.bears.requirements.DistributionRequirement.**DistributionRequirement**(***manager_comma*

Bases: *coalib.bears.requirements.PackageRequirement.PackageRequirement*

This class is a subclass of `PackageRequirement`. It specifices the proper type automatically.

coalib.bears.requirements.GemRequirement module

class coalib.bears.requirements.GemRequirement.**GemRequirement**(*package*, *version=''*, *require=''*)

Bases: *coalib.bears.requirements.PackageRequirement.PackageRequirement*

This class is a subclass of `PackageRequirement`. It specifies the proper type for `ruby` packages automatically and provide a function to check for the requirement.

is_installed()

Checks if the dependency is installed.

Parameters **return** – True if dependency is installed, false otherwise.

coalib.bears.requirements.GoRequirement module

class coalib.bears.requirements.GoRequirement.**GoRequirement**(*package*, *version=''*, *flag=''*)

Bases: *coalib.bears.requirements.PackageRequirement.PackageRequirement*

This class is a subclass of `PackageRequirement`. It specifies the proper type for `go` packages automatically and provide a function to check for the requirement.

is_installed()

Checks if the dependency is installed.

Parameters **return** – True if dependency is installed, false otherwise.

coalib.bears.requirements.JuliaRequirement module

class coalib.bears.requirements.JuliaRequirement.**JuliaRequirement**(*package*, *version=''*)

Bases: *coalib.bears.requirements.PackageRequirement.PackageRequirement*

This class is a subclass of `PackageRequirement`. It specifies the proper type for `julia` packages automatically and provide a function to check for the requirement.

is_installed()

Checks if the dependency is installed.

Returns `True` if dependency is installed, `False` otherwise.

coalib.bears.requirements.NpmRequirement module

class coalib.bears.requirements.NpmRequirement.**NpmRequirement**(*package*, *version=''*)

Bases: *coalib.bears.requirements.PackageRequirement.PackageRequirement*

This class is a subclass of `PackageRequirement`. It specifies the proper type for `npm` packages automatically and provide a function to check for the requirement.

is_installed()

Checks if the dependency is installed.

Parameters **return** – True if dependency is installed, false otherwise.

coalib.bears.requirements.PackageRequirement module

class coalib.bears.requirements.PackageRequirement.**PackageRequirement**(*type: str*, *package: str*, *version=''*)

Bases: `object`

This class helps keeping track of bear requirements. It should simply be appended to the REQUIREMENTS tuple inside the Bear class.

Two `PackageRequirements` should always be equal if they have the same type, package and version:

```
>>> pr1 = PackageRequirement('pip', 'coala_decorators', '0.1.0')
>>> pr2 = PackageRequirement('pip', 'coala_decorators', '0.1.0')
>>> pr1 == pr2
True
```

is_installed()

Check if the requirement is satisfied.

```
>>> PackageRequirement('pip',                                    'coala_decorators
 ↪',                                  '0.2.1').is_installed()
Traceback (most recent call last):
...
NotImplementedError
```

Returns Returns True if satisfied, False if not.

coalib.bears.requirements.PipRequirement module

class coalib.bears.requirements.PipRequirement.**PipRequirement**(*package*, *version=''*)
Bases: *coalib.bears.requirements.PackageRequirement.PackageRequirement*

This class is a subclass of `PackageRequirement`. It specifies the proper type for `python` packages automatically and provide a function to check for the requirement.

is_installed()
Checks if the dependency is installed.

Parameters **return** – True if dependency is installed, false otherwise.

coalib.bears.requirements.RscriptRequirement module

class coalib.bears.requirements.RscriptRequirement.**RscriptRequirement**(*package*, *version=''*, *flag=''*, *repo=''*)
Bases: *coalib.bears.requirements.PackageRequirement.PackageRequirement*

This class is a subclass of `PackageRequirement`. It specifies the proper type for `R` packages automatically and provide a function to check for the requirement.

is_installed()
Checks if the dependency is installed.

Parameters **return** – True if dependency is installed, false otherwise.

Module contents

Submodules

coalib.bears.BEAR_KIND module

coalib.bears.Bear module

class coalib.bears.Bear.**Bear**(*section: coalib.settings.Section.Section*, *message_queue*, *timeout=0*)
Bases: pyprint.Printer.Printer, *coalib.output.printers.LogPrinter.LogPrinterMixin*

A bear contains the actual subroutine that is responsible for checking source code for certain specifications. However it can actually do whatever it wants with the files it gets. If you are missing some Result type, feel free to contact us and/or help us extending the coalib.

This is the base class for every bear. If you want to write an bear, you will probably want to look at the GlobalBear and LocalBear classes that inherit from this class. In any case you'll want to overwrite at least the run method. You can send debug/warning/error messages through the debug(), warn(), err() functions. These will send the appropriate messages so that they are outputted. Be aware that if you use err(), you are expected to also terminate the bear run-through immediately.

If you need some setup or teardown for your bear, feel free to overwrite the set_up() and tear_down() functions. They will be invoked before/after every run invocation.

Settings are available at all times through self.section.

To indicate which languages your bear supports, just give it the `LANGUAGES` value which should be a set of string(s):

```
>>> class SomeBear(Bear):
...     LANGUAGES = {'C', 'CPP','C#', 'D'}
```

To indicate the requirements of the bear, assign `REQUIREMENTS` a set with instances of `PackageRequirements`.

```
>>> class SomeBear(Bear):
...     REQUIREMENTS = {
...         PackageRequirement('pip', 'coala_decorators', '0.2.1')}
```

If your bear uses requirements from a manager we have a subclass from, you can use the subclass, such as `PipRequirement`, without specifying manager:

```
>>> class SomeBear(Bear):
...     REQUIREMENTS = {PipRequirement('coala_decorators', '0.2.1')}
```

To specify additional attributes to your bear, use the following:

```
>>> class SomeBear(Bear):
...     AUTHORS = {'Jon Snow'}
...     AUTHORS_EMAILS = {'jon_snow@gmail.com'}
...     MAINTAINERS = {'Catelyn Stark'}
...     MAINTAINERS_EMAILS = {'catelyn_stark@gmail.com'}
...     LICENSE = 'AGPL-3.0'
...     ASCIINEMA_URL = 'https://asciinema.org/a/80761'
```

If the maintainers are the same as the authors, they can be omitted:

```
>>> class SomeBear(Bear):
...     AUTHORS = {'Jon Snow'}
...     AUTHORS_EMAILS = {'jon_snow@gmail.com'}
>>> SomeBear.maintainers
{'Jon Snow'}
>>> SomeBear.maintainers_emails
{'jon_snow@gmail.com'}
```

If your bear needs to include local files, then specify it giving strings containing relative file paths to the INCLUDE_LOCAL_FILES set:

```
>>> class SomeBear(Bear):
...     INCLUDE_LOCAL_FILES = {'checkstyle.jar', 'google_checks.xml'}
```

To keep track easier of what a bear can do, simply tell it to the CAN_FIX and the CAN_DETECT sets. Possible values:

```
>>> CAN_DETECT = {'Syntax', 'Formatting', 'Security', 'Complexity', 'Smell',
... 'Unused Code', 'Redundancy', 'Variable Misuse', 'Spelling',
... 'Memory Leak', 'Documentation', 'Duplication', 'Commented Code',
... 'Grammar', 'Missing Import', 'Unreachable Code', 'Undefined Element',
... 'Code Simplification'}
>>> CAN_FIX = {'Syntax', ...}
```

Specifying something to CAN_FIX makes it obvious that it can be detected too, so it may be omitted:

```
>>> class SomeBear(Bear):
...     CAN_DETECT = {'Syntax', 'Security'}
...     CAN_FIX = {'Redundancy'}
>>> list(sorted(SomeBear.can_detect))
['Redundancy', 'Security', 'Syntax']
```

Every bear has a data directory which is unique to that particular bear:

```
>>> class SomeBear(Bear): pass
>>> class SomeOtherBear(Bear): pass
>>> SomeBear.data_dir == SomeOtherBear.data_dir
False
```

BEAR_DEPS contains bear classes that are to be executed before this bear gets executed. The results of these bears will then be passed to the run method as a dict via the dependency_results argument. The dict will have the name of the Bear as key and the list of its results as results:

```
>>> class SomeBear(Bear): pass
>>> class SomeOtherBear(Bear):
...     BEAR_DEPS = {SomeBear}
>>> SomeOtherBear.BEAR_DEPS
{<class 'coalib.bears.Bear.SomeBear'>}
```

ASCIINEMA_URL = ‘’

AUTHORS = set()

AUTHORS_EMAILS = set()

BEAR_DEPS = set()

CAN_DETECT = set()

CAN_FIX = set()

INCLUDE_LOCAL_FILES = set()

LANGUAGES = set()

LICENSE = ‘’

MAINTAINERS = set()

MAINTAINERS_EMAILS = set()

PLATFORMS = {‘any’}

REQUIREMENTS = set()

can_detect = set()

classmethod check_prerequisites()

Checks whether needed runtime prerequisites of the bear are satisfied.

This function gets executed at construction and returns True by default.

Section value requirements shall be checked inside the `run` method.

Returns True if prerequisites are satisfied, else False or a string that serves a more detailed description of what’s missing.

data_dir = ‘/home/docs/.local/share/coala-bears/Bear’

download_cached_file(*url*, *filename*)
Downloads the file if needed and caches it for the next time. If a download happens, the user will be informed.

Take a sane simple bear:

```
>>> from queue import Queue
>>> bear = Bear(Section("a section"), Queue())
```

We can now carelessly query for a neat file that doesn't exist yet:

```
>>> from os import remove
>>> if exists(join(bear.data_dir, "a_file")):
...     remove(join(bear.data_dir, "a_file"))
>>> file = bear.download_cached_file("http://gitmate.com/", "a_file")
```

If we download it again, it'll be much faster as no download occurs:

```
>>> newfile = bear.download_cached_file("http://gitmate.com/", "a_file")
>>> newfile == file
True
```

Parameters

- **url** – The URL to download the file from.
- **filename** – The filename it should get, e.g. "test.txt".

Returns A full path to the file ready for you to use!

execute(**args*, ***kwargs*)

get_config_dir()
Gives the directory where the configuration file is

Returns Directory of the config file

classmethod **get_metadata**()

Returns Metadata for the run function. However parameters like `self` or parameters implicitly used by coala (e.g. filename for local bears) are already removed.

classmethod **get_non_optional_settings**()
This method has to determine which settings are needed by this bear. The user will be prompted for needed settings that are not available in the settings file so don't include settings where a default value would do.

Returns A dictionary of needed settings as keys and a tuple of help text and annotation as values

static **kind**()

Returns The kind of the bear

log_message(*log_message*, *timestamp=None*, ***kwargs*)

maintainers = set()

maintainers_emails = set()

classmethod **missing_dependencies**(*lst*)
Checks if the given list contains all dependencies.

Parameters **lst** – A list of all already resolved bear classes (not instances).

Returns A set of missing dependencies.

name = 'Bear'

new_result
Returns a partial for creating a result with this bear already bound.

run (**args*, ***, *dependency_results=None*, ***kwargs*)

run_bear_from_section (*args*, *kwargs*)

static **setup_dependencies** ()
This is a user defined function that can download and set up dependencies (via download_cached_file or arbitrary other means) in an OS independent way.

coalib.bears.GlobalBear module

class coalib.bears.GlobalBear.**GlobalBear** (*file_dict*, *section*, *message_queue*, *timeout=0*)
Bases: *coalib.bears.Bear.Bear*

A GlobalBear is able to analyze semantic facts across several file.

The results of a GlobalBear will be presented grouped by the origin Bear. Therefore Results spanning above multiple files are allowed and will be handled right.

If you only look at one file at once anyway a LocalBear is better for your needs. (And better for performance and usability for both user and developer.)

static **kind** ()

run (**args*, ***, *dependency_results=None*, ***kwargs*)
Handles all files in file_dict.

Returns A list of Result type.

coalib.bears.LocalBear module

class coalib.bears.LocalBear.**LocalBear** (*section: coalib.settings.Section.Section*, *message_queue*, *timeout=0*)
Bases: *coalib.bears.Bear.Bear*

A LocalBear is a Bear that analyzes only one file at once. It therefore can not analyze semantical facts over multiple files.

This has the advantage that it can be highly parallelized. In addition, the results from multiple bears for one file can be shown together for that file, which is better to grasp for the user. coala takes care of all that.

Examples for LocalBear's could be:

- A SpaceConsistencyBear that checks every line for trailing whitespaces, tabs, etc.
- A VariableNameBear that checks variable names and constant names for certain conditions

classmethod **get_metadata** ()

static **kind** ()

run (*filename*, *file*, **args*, ***, *dependency_results=None*, ***kwargs*)
Handles the given file.

Parameters

- **filename** – The filename of the file
- **file** – The file contents as string array

Returns A list of Result

Module contents

1.1.3 coalib.collecting package

Submodules

coalib.collecting.Collectors module

coalib.collecting.Collectors.**collect_all_bears_from_sections**(*sections*, *log_printer*)

Collect all kinds of bears from bear directories given in the sections.

Parameters

- **sections** – List of sections so bear_dirs are taken into account
- **log_printer** – Log_printer to handle logging

Returns Tuple of dictionaries of local and global bears. The dictionary key is section class and dictionary value is a list of Bear classes

coalib.collecting.Collectors.**collect_bears**(*bear_dirs*, *bear_globs*, *kinds*, *log_printer*, *warn_if_unused_glob=True*)

Collect all bears from bear directories that have a matching kind matching the given globs.

Parameters

- **bear_dirs** – Directory name or list of such that can contain bears.
- **bear_globs** – Globs of bears to collect.
- **kinds** – List of bear kinds to be collected.
- **log_printer** – log_printer to handle logging.
- **warn_if_unused_glob** – True if warning message should be shown if a glob didn't give any bears.

Returns Tuple of list of matching bear classes based on kind. The lists are in the same order as kinds.

coalib.collecting.Collectors.**collect_dirs**(*dir_paths*, *ignored_dir_paths=None*)

Evaluate globs in directory paths and return all matching directories

Parameters

- **dir_paths** – File path or list of such that can include globs
- **ignored_dir_paths** – List of globs that match to-be-ignored dirs

Returns List of paths of all matching directories

coalib.collecting.Collectors.**collect_files**(*file_paths*, *log_printer*, *ignored_file_paths=None*, *limit_file_paths=None*)

Evaluate globs in file paths and return all matching files

Parameters

- **file_paths** – File path or list of such that can include globs
- **ignored_file_paths** – List of globs that match to-be-ignored files

- **limit_file_paths** – List of globs that the files are limited to

Returns List of paths of all matching files

coalib.collecting.Collectors.**collect_registered_bears_dirs**(*entrypoint*)
Searches setuptools for the entrypoint and returns the bear directories given by the module.

Parameters **entrypoint** – The entrypoint to find packages with.

Returns List of bear directories.

coalib.collecting.Collectors.**filter_capabilities_by_languages**(*bears*, *languages*)
Filters the bears capabilities by languages.

Parameters

- **bears** – Dictionary with sections as keys and list of bears as values.
- **languages** – Languages that bears are being filtered on.

Returns New dictionary with languages as keys and their bears capabilities as values. The capabilities are stored in a tuple of two elements where the first one represents what the bears can detect, and the second one what they can fix.

coalib.collecting.Collectors.**filter_section_bears_by_languages**(*bears*, *languages*)
Filters the bears by languages.

Parameters

- **bears** – The dictionary of the sections as keys and list of bears as values.
- **languages** – Languages that bears are being filtered on.

Returns New dictionary with filtered out bears that don't match any language from languages.

coalib.collecting.Collectors.**get_all_bears_names**()

coalib.collecting.Collectors.**icollect**(*file_paths*, *ignored_globs=None*)
Evaluate globs in file paths and return all matching files.

Parameters

- **file_paths** – File path or list of such that can include globs
- **ignored_globs** – List of globs to ignore when matching files

Returns Iterator that yields tuple of path of a matching file, the glob where it was found

coalib.collecting.Collectors.**icollect_bears**(*bear_dir_glob*, *bear_globs*, *kinds*, *log_printer*)
Collect all bears from bear directories that have a matching kind.

Parameters

- **bear_dir_glob** – Directory globs or list of such that can contain bears
- **bear_globs** – Globs of bears to collect
- **kinds** – List of bear kinds to be collected
- **log_printer** – Log_printer to handle logging

Returns Iterator that yields a tuple with bear class and which bear_glob was used to find that bear class.

coalib.collecting.Dependencies module

exception coalib.collecting.Dependencies.**CircularDependencyError**
Bases: Exception

classmethod for_bears(*bears*)
Creates the CircularDependencyError with a helpful message about the dependency.

coalib.collecting.Dependencies.**resolve**(*bears*)
Collects all dependencies of the given bears. This will also remove duplicates.

Parameters **bears** – The given bears. Will not be modified.

Returns The new list of bears, sorted so that it can be executed sequentially without dependency issues.

coalib.collecting.Importers module

coalib.collecting.Importers.**iimport_objects**(*file_paths*, *names=None*, *types=None*, *supers=None*, *attributes=None*, *local=False*, *suppress_output=False*)
Import all objects from the given modules that fulfill the requirements

Parameters

- **file_paths** – File path(s) from which objects will be imported.
- **names** – Name(s) an objects need to have one of.
- **types** – Type(s) an objects need to be out of.
- **supers** – Class(es) objects need to be a subclass of.
- **attributes** – Attribute(s) an object needs to (all) have.
- **local** – If True: Objects need to be defined in the file they appear in to be collected.
- **suppress_output** – Whether console output from stdout shall be suppressed or not.

Returns An iterator that yields all matching python objects.

Raises **Exception** – Any exception that is thrown in module code or an ImportError if paths are erroneous.

coalib.collecting.Importers.**import_objects**(*file_paths*, *names=None*, *types=None*, *supers=None*, *attributes=None*, *local=False*, *verbose=False*)
Import all objects from the given modules that fulfill the requirements

Parameters

- **file_paths** – File path(s) from which objects will be imported
- **names** – Name(s) an objects need to have one of
- **types** – Type(s) an objects need to be out of
- **supers** – Class(es) objects need to be a subclass of
- **attributes** – Attribute(s) an object needs to (all) have
- **local** – if True: Objects need to be defined in the file they appear in to be collected

Returns list of all matching python objects

Raises **Exception** – Any exception that is thrown in module code or an ImportError if paths are erroneous.

coalib.collecting.Importers.**object_defined_in**(*obj*, *file_path*)

Check if the object is defined in the given file.

```
>>> object_defined_in(object_defined_in, __file__)
True
>>> object_defined_in(object_defined_in, "somewhere else")
False
```

Builtins are always defined outside any given file:

```
>>> object_defined_in(False, __file__)
False
```

Parameters

- **obj** – The object to check.
- **file_path** – The path it might be defined in.

Returns True if the object is defined in the file.

Module contents

1.1.4 coalib.misc package

Submodules

coalib.misc.Annotations module

coalib.misc.Annotations.**typechain**(**args*)

Returns function which applies the first transformation it can from args and returns transformed value, or the value itself if it is in args.

```
>>> function = typechain(int, 'a', ord, None)
>>> function("10")
10
>>> function("b")
98
>>> function("a")
'a'
>>> function(int)
<class 'int'>
>>> function(None) is None
True
>>> function("str")
Traceback (most recent call last):
    ...
ValueError: Couldn't convert value 'str' to any specified type or find it in␣
 ↪specified values.
```

Raises **TypeError** – Raises when either no functions are specified for checking.

coalib.misc.BuildManPage module

class coalib.misc.BuildManPage.**BuildManPage**(*dist*)
Bases: distutils.cmd.Command

Add a build_manpage command to your setup.py. To use this Command class add a command to call this class:

```
# For setuptools
setup(
      entry_points={
        "distutils.commands": [
            "build_manpage = coalib.misc.BuildManPage:BuildManPage"
        ]
      }
)

# For distutils
from coalib.misc.BuildManPage import BuildManPage
setup(
      cmdclass={'build_manpage': BuildManPage}
)
```

You can then use the following setup command to produce a man page:

```
$ python setup.py build_manpage --output=coala.1             --parser=coalib.
↪parsing.DefaultArgParser:default_arg_parser
```

If automatically want to build the man page every time you invoke your build, add to your `setup.cfg` the following:

```
[build_manpage]
output = <appname>.1
parser = <path_to_your_parser>
```

finalize_options()

initialize_options()

run()

user_options = [('output=', 'O', 'output file'), ('parser=', None, 'module path to an ArgumentParser instance(e.g. my

class coalib.misc.BuildManPage.**ManPageFormatter**(*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*, *desc=None*, *long_desc=None*, *ext_sections=None*, *parser=None*)
Bases: argparse.HelpFormatter

format_man_page()

coalib.misc.Caching module

class coalib.misc.Caching.**FileCache**(*log_printer: coalib.output.printers.LogPrinter.LogPrinterMixin*, *project_dir: str*, *flush_cache: bool = False*)
Bases: object

This object is a file cache that helps in collecting only the changed and new files since the last run. Example/Tutorial:

```
>>> from pyprint.NullPrinter import NullPrinter
>>> from coalib.output.printers.LogPrinter import LogPrinter
>>> import logging
>>> import copy, time
>>> log_printer = LogPrinter()
>>> log_printer.log_level = logging.CRITICAL
```

To initialize the cache create an instance for the project:

```
>>> cache = FileCache(log_printer, "test", flush_cache=True)
```

Now we can track new files by running:

```
>>> cache.track_files(["a.c", "b.c"])
```

Since all cache operations are lazy (for performance), we need to explicitly write the cache to disk for persistence in future uses: (Note: The cache will automatically figure out the write location)

```
>>> cache.write()
```

Let's go into the future:

```
>>> time.sleep(1)
```

Let's create a new instance to simulate a separate run:

```
>>> cache = FileCache(log_printer, "test", flush_cache=False)
```

```
>>> old_data = copy.deepcopy(cache.data)
```

We can mark a file as changed by doing:

```
>>> cache.untrack_files({"a.c"})
```

Again write to disk after calculating the new cache times for each file:

```
>>> cache.write()
>>> new_data = cache.data
```

Since 'a.c' as a changed file:

```
>>> "a.c" not in cache.data
True
>>> "a.c" in old_data
True
```

Since 'b.c' was untouched after the second run, its time was updated to the latest value:

```
>>> old_data["b.c"] < new_data["b.c"]
True
```

flush_cache()
Flushes the cache and deletes the relevant file.

get_uncached_files(*files*)
Returns the set of files that are not in the cache yet or have been untracked.

Parameters **files** – The list of collected files.

Returns A set of files that are uncached.

track_files(*files*)
Start tracking files given in `files` by adding them to the database.

Parameters **files** – A set of files that need to be tracked. These files are initialized with their last modified tag as -1.

untrack_files(*files*)
Removes the given files from the cache so that they are no longer considered cached for this and the next run.

Parameters **files** – A set of files to remove from cache.

write()
Update the last run time on the project for each file to the current time. Using this object as a contextmanager is preferred (that will automatically call this method on exit).

coalib.misc.CachingUtilities module

coalib.misc.CachingUtilities.**delete_files**(*log_printer*, *identifiers*)
Delete the given identifiers from the user's coala data directory.

Parameters

- **log_printer** – A LogPrinter object to use for logging.
- **identifiers** – The list of files to be deleted.

Returns True if all the given files were successfully deleted. False otherwise.

coalib.misc.CachingUtilities.**get_data_path**(*log_printer*, *identifier*)
Get the full path of `identifier` present in the user's data directory.

Parameters

- **log_printer** – A LogPrinter object to use for logging.
- **identifier** – The file whose path needs to be expanded.

Returns Full path of the file, assuming it's present in the user's config directory. Returns `None` if there is a `PermissionError` in creating the directory.

coalib.misc.CachingUtilities.**get_settings_hash**(*sections*, *targets=[]*, *ignore_settings: list = ['disable_caching']*)
Compute and return a unique hash for the settings.

Parameters

- **sections** – A dict containing the settings for each section.
- **targets** – The list of sections that are enabled.
- **ignore_settings** – Setting keys to remove from sections before hashing.

Returns A MD5 hash that is unique to the settings used.

coalib.misc.CachingUtilities.**hash_id**(*text*)
Hashes the given text.

Parameters **text** – String to to be hashed

Returns A MD5 hash of the given string

coalib.misc.CachingUtilities.**pickle_dump**(*log_printer*, *identifier*, *data*)
Write `data` into the file `filename` present in the user config directory.

Parameters

- **log_printer** – A LogPrinter object to use for logging.
- **identifier** – The name of the file present in the user config directory.
- **data** – Data to be serialized and written to the file using pickle.

Returns True if the write was successful. False if there was a permission error in writing.

coalib.misc.CachingUtilities.**pickle_load**(*log_printer*, *identifier*, *fallback=None*)
Get the data stored in `filename` present in the user config directory. Example usage:

```
>>> from pyprint.NullPrinter import NullPrinter
>>> from coalib.output.printers.LogPrinter import LogPrinter
>>> log_printer = LogPrinter(NullPrinter())
>>> test_data = {"answer": 42}
>>> pickle_dump(log_printer, "test_project", test_data)
True
>>> pickle_load(log_printer, "test_project")
{'answer': 42}
>>> pickle_load(log_printer, "nonexistent_project")
>>> pickle_load(log_printer, "nonexistent_project", fallback=42)
42
```

Parameters

- **log_printer** – A LogPrinter object to use for logging.
- **identifier** – The name of the file present in the user config directory.
- **fallback** – Return value to fallback to in case the file doesn't exist.

Returns Data that is present in the file, if the file exists. Otherwise the `default` value is returned.

coalib.misc.CachingUtilities.**settings_changed**(*log_printer*, *settings_hash*)
Determine if the settings have changed since the last run with caching.

Parameters

- **log_printer** – A LogPrinter object to use for logging.
- **settings_hash** – A MD5 hash that is unique to the settings used.

Returns Return True if the settings hash has changed Return False otherwise.

coalib.misc.CachingUtilities.**update_settings_db**(*log_printer*, *settings_hash*)
Update the config file last modification date.

Parameters

- **log_printer** – A LogPrinter object to use for logging.
- **settings_hash** – A MD5 hash that is unique to the settings used.

coalib.misc.Compatibility module

coalib.misc.Constants module

coalib.misc.Constants.**configure_logging**()
Configures the logging with hard coded dictionary.

coalib.misc.ContextManagers module

coalib.misc.ContextManagers.**change_directory**(*path*)

coalib.misc.ContextManagers.**make_temp**(*suffix=''*, *prefix='tmp'*, *dir=None*)
Creates a temporary file with a closed stream and deletes it when done.

Returns A contextmanager retrieving the file path.

coalib.misc.ContextManagers.**prepare_file**(*lines*, *filename*, *force_linebreaks=True*, *create_tempfile=True*, *tempfile_kwargs={}*)
Can create a temporary file (if filename is None) with the lines. Can also add a trailing newline to each line specified if needed.

Parameters

- **lines** – The lines from the file. (list or tuple of strings)
- **filename** – The filename to be prepared.
- **force_linebreaks** – Whether to append newlines at each line if needed.
- **create_tempfile** – Whether to save lines in tempfile if needed.
- **tempfile_kwargs** – Kwargs passed to tempfile.mkstemp().

coalib.misc.ContextManagers.**replace_stderr**(*replacement*)
Replaces stderr with the replacement, yields back to the caller and then reverts everything back.

coalib.misc.ContextManagers.**replace_stdout**(*replacement*)
Replaces stdout with the replacement, yields back to the caller and then reverts everything back.

coalib.misc.ContextManagers.**retrieve_stderr**()
Yields a StringIO object from which one can read everything that was printed to stderr. (It won't be printed to the real stderr!)

Example usage:

with retrieve_stderr() as stderr: print("something") # Won't print to the console what_was_printed = stderr.getvalue() # Save the value

coalib.misc.ContextManagers.**retrieve_stdout**()
Yields a StringIO object from which one can read everything that was printed to stdout. (It won't be printed to the real stdout!)

Example usage:

with retrieve_stdout() as stdout: print("something") # Won't print to the console what_was_printed = stdout.getvalue() # Save the value

coalib.misc.ContextManagers.**simulate_console_inputs**(**inputs*)
Does some magic to simulate the given inputs to any calls to the `input` builtin. This yields back an InputGenerator object so you can check which input was already used and append any additional inputs you want. Example:

with simulate_console_inputs(0, 1, 2) as generator: assert(input() == 0) assert(generator.last_input == 0) generator.inputs.append(3) assert(input() == 1) assert(input() == 2) assert(input() == 3) assert(generator.last_input == 3)

Parameters **inputs** – Any inputs to simulate.

Raises **ValueError** – Raised when was asked for more input but there's no more provided.

coalib.misc.ContextManagers.**subprocess_timeout**(*sub_process*, *seconds*, *kill_pg=False*)
Kill subprocess if the sub process takes more the than the timeout.

Parameters

- **sub_process** – The sub process to run.
- **seconds** – The number of seconds to allow the test to run for. If set to 0 or a negative value, it waits indefinitely. Floats can be used to specify units smaller than seconds.
- **kill_pg** – Boolean whether to kill the process group or only this process. (not applicable for windows)

coalib.misc.ContextManagers.**suppress_stdout**()
Suppresses everything going to stdout.

coalib.misc.DictUtilities module

coalib.misc.DictUtilities.**add_pair_to_dict**(*key*, *value*, *dictionary*)
Add (key, value) pair to the dictionary. The value is added to a list of values for the key.

coalib.misc.DictUtilities.**inverse_dicts**(**dicts*)
Inverts the dicts, e.g. {1: 2, 3: 4} and {2: 3, 4: 4} will be inverted {2: [1], 3: [2], 4: [3, 4]}. This also handles dictionaries with Iterable items as values e.g. {1: [1, 2, 3], 2: [3, 4, 5]} and {2: [1], 3: [2], 4: [3, 4]} will be inverted to {1: [1, 2], 2: [1, 3], 3: [1, 2, 4], 4: [2, 4], 5: [2]}. No order is preserved.

Parameters **dicts** – The dictionaries to invert.

Returns The inversed dictionary which merges all dictionaries into one.

coalib.misc.DictUtilities.**update_ordered_dict_key**(*dictionary*, *old_key*, *new_key*)

coalib.misc.Enum module

coalib.misc.Enum.**enum**(**sequential*, ***named*)

coalib.misc.Exceptions module

coalib.misc.Exceptions.**get_exitcode**(*exception*, *log_printer=None*)

coalib.misc.MutableValue module

class coalib.misc.MutableValue.**MutableValue**(*val=None*)
Bases: object

coalib.misc.Shell module

coalib.misc.Shell.**call_without_output = functools.partial(<function call>, stdout=-3, stderr=-3)**
Uses subprocess.call to execute a command, but suppresses the output and the errors.

coalib.misc.Shell.**get_shell_type**()
Finds the current shell type based on the outputs of common pre-defined variables in them. This is useful to identify which sort of escaping is required for strings.

Returns The shell type. This can be either "powershell" if Windows Powershell is detected, "cmd" if command prompt is been detected or "sh" if it's neither of these.

coalib.misc.Shell.**run_interactive_shell_command**(*command*, ***kwargs*)
Runs a single command in shell and provides stdout, stderr and stdin streams.

This function creates a context manager that sets up the process (using `subprocess.Popen()`), returns to caller and waits for process to exit on leaving.

By default the process is opened in `universal_newlines` mode and creates pipes for all streams (stdout, stderr and stdin) using `subprocess.PIPE` special value. These pipes are closed automatically, so if you want to get the contents of the streams you should retrieve them before the context manager exits.

```
>>> with run_interactive_shell_command(["echo", "TEXT"]) as p:
...     stdout = p.stdout
...     stdout_text = stdout.read()
>>> stdout_text
'TEXT\n'
>>> stdout.closed
True
```

Custom streams provided are not closed except of `subprocess.PIPE`.

```
>>> from tempfile import TemporaryFile
>>> stream = TemporaryFile()
>>> with run_interactive_shell_command(["echo", "TEXT"],
...                                    stdout=stream) as p:
...     stderr = p.stderr
>>> stderr.closed
True
>>> stream.closed
False
```

Parameters

- **command** – The command to run on shell. This parameter can either be a sequence of arguments that are directly passed to the process or a string. A string gets splitted beforehand using `shlex.split()`. If providing `shell=True` as a keyword-argument, no `shlex.split()` is performed and the command string goes directly to `subprocess.Popen()`.
- **kwargs** – Additional keyword arguments to pass to `subprocess.Popen` that are used to spawn the process.

Returns A context manager yielding the process started from the command.

coalib.misc.Shell.**run_shell_command**(*command*, *stdin=None*, ***kwargs*)
Runs a single command in shell and returns the read stdout and stderr data.

This function waits for the process (created using `subprocess.Popen()`) to exit. Effectively it wraps `run_interactive_shell_command()` and uses `communicate()` on the process.

See also `run_interactive_shell_command()`.

Parameters

- **`command`** – The command to run on shell. This parameter can either be a sequence of arguments that are directly passed to the process or a string. A string gets splitted beforehand using `shlex.split()`.
- **`stdin`** – Initial input to send to the process.
- **`kwargs`** – Additional keyword arguments to pass to `subprocess.Popen` that is used to spawn the process.

Returns A tuple with `(stdoutstring,stderrstring)`.

Module contents

1.1.5 coalib.output package

Subpackages

coalib.output.printers package

Submodules

coalib.output.printers.LOG_LEVEL module

coalib.output.printers.ListLogPrinter module

class coalib.output.printers.ListLogPrinter.**ListLogPrinter**(*log_level=30*, *timestamp_format='%X'*)

Bases: pyprint.Printer.Printer, *coalib.output.printers.LogPrinter.LogPrinterMixin*

A ListLogPrinter is a log printer which collects all LogMessages to a list so that the logs can be used at a later time.

`log_message`(*log_message*, ***kwargs*)

coalib.output.printers.LogPrinter module

class coalib.output.printers.LogPrinter.**LogPrinter**(*printer=None*, *log_level=10*, *timestamp_format='%X'*)

Bases: *coalib.output.printers.LogPrinter.LogPrinterMixin*

This class is deprecated and will be soon removed. To get logger use logging.getLogger(__name__). Make sure that you're getting it when the logging configuration is loaded.

The LogPrinter class allows to print log messages to an underlying Printer.

This class is an adapter, means you can create a LogPrinter from every existing Printer instance.

`log_level`
Returns current log_level used in logger.

`log_message`(*log_message*, ***kwargs*)

printer
Returns the underlying printer where logs are printed to.

class coalib.output.printers.LogPrinter.**LogPrinterMixin**
Bases: object

Provides access to the logging interfaces (e.g. err, warn, info) by routing them to the log_message method, which should be implemented by descendants of this class.

debug(**messages*, ***, *delimiter=' '*, *timestamp=None*, ***kwargs*)

err(**messages*, ***, *delimiter=' '*, *timestamp=None*, ***kwargs*)

info(**messages*, ***, *delimiter=' '*, *timestamp=None*, ***kwargs*)

log(*log_level*, *message*, *timestamp=None*, ***kwargs*)

log_exception(*message*, *exception*, *log_level=40*, *timestamp=None*, ***kwargs*)
If the log_level of the printer is greater than DEBUG, it prints only the message. If it is DEBUG or lower, it shows the message along with the traceback of the exception.

Parameters

- **message** – The message to print.
- **exception** – The exception to print.
- **log_level** – The log_level of this message (not used when logging the traceback. Tracebacks always have a level of DEBUG).
- **timestamp** – The time at which this log occurred. Defaults to the current time.
- **kwargs** – Keyword arguments to be passed when logging the message (not used when logging the traceback).

log_message(*log_message*, ***kwargs*)
It is your reponsibility to implement this method, if you're using this mixin.

warn(**messages*, ***, *delimiter=' '*, *timestamp=None*, ***kwargs*)

Module contents

This package holds printer objects. Printer objects are general purpose and not tied to coala.

If you need logging capabilities please take a look at the LogPrinter object which adds logging capabilities "for free" if used as base class for any other printer.

Submodules

coalib.output.ConfWriter module

class coalib.output.ConfWriter.**ConfWriter**(*file_name*, *key_value_delimiters=('=',)*, *comment_separators=('#',)*, *key_delimiters=(',', ' ')*, *section_name_surroundings=mappingproxy({'[': ']'})*, *section_override_delimiters=('.',)*, *unsavable_keys=('save',)*)
Bases: pyprint.ClosableObject.ClosableObject

static is_comment(*key*)

write_section(*section*)

write_sections(*sections*)

coalib.output.ConsoleInteraction module

class coalib.output.ConsoleInteraction.**BackgroundMessageStyle**
Bases: pygments.style.Style

styles = {Token.Name.Function: '', Token.Comment.Hashbang: '', Token.Keyword.Constant: '', Token.Name.Property

class coalib.output.ConsoleInteraction.**BackgroundSourceRangeStyle**
Bases: pygments.style.Style

styles = {Token.Name.Function: '', Token.Comment.Hashbang: '', Token.Keyword.Constant: '', Token.Name.Property

class coalib.output.ConsoleInteraction.**NoColorStyle**
Bases: pygments.style.Style

styles = {Token.Name.Function: '', Token.Comment.Hashbang: '', Token.Keyword.Constant: '', Token.Name.Property

coalib.output.ConsoleInteraction.**acquire_actions_and_apply**(*console_printer*, *section*, *file_diff_dict*, *result*, *file_dict*, *cli_actions=None*)

Acquires applicable actions and applies them.

Parameters

- **console_printer** – Object to print messages on the console.
- **section** – Name of section to which the result belongs.
- **file_diff_dict** – Dictionary containing filenames as keys and Diff objects as values.
- **result** – A derivative of Result.
- **file_dict** – A dictionary containing all files with filename as key.
- **cli_actions** – The list of cli actions available.

coalib.output.ConsoleInteraction.**acquire_settings**(*log_printer*, *settings_names_dict*, *section*)

This method prompts the user for the given settings.

Parameters

- **log_printer** – Printer responsible for logging the messages. This is needed to comply with the interface.
- **settings_names_dict** – A dictionary with the settings name as key and a list containing a description in [0] and the name of the bears who need this setting in [1] and following.

  Example:

```
{"UseTabs": ["describes whether tabs should be used instead of spaces",
             "SpaceConsistencyBear",
             "SomeOtherBear"]}
```

Parameters **section** – The section the action corresponds to.

Returns A dictionary with the settings name as key and the given value as value.

coalib.output.ConsoleInteraction.**ask_for_action_and_apply**(*console_printer*, *section*, *metadata_list*, *action_dict*, *failed_actions*, *result*, *file_diff_dict*, *file_dict*)

Asks the user for an action and applies it.

Parameters

- **console_printer** – Object to print messages on the console.
- **section** – Currently active section.
- **metadata_list** – Contains metadata for all the actions.
- **action_dict** – Contains the action names as keys and their references as values.
- **failed_actions** – A set of all actions that have failed. A failed action remains in the list until it is successfully executed.
- **result** – Result corresponding to the actions.
- **file_diff_dict** – If it is an action which applies a patch, this contains the diff of the patch to be applied to the file with filename as keys.
- **file_dict** – Dictionary with filename as keys and its contents as values.

Returns Returns a boolean value. True will be returned, if it makes sense that the user may choose to execute another action, False otherwise.

coalib.output.ConsoleInteraction.**choose_action**(*console_printer*, *actions*)

Presents the actions available to the user and takes as input the action the user wants to choose.

Parameters

- **console_printer** – Object to print messages on the console.
- **actions** – Actions available to the user.

Returns Return choice of action of user.

coalib.output.ConsoleInteraction.**format_lines**(*lines*, *line_nr=''*)

coalib.output.ConsoleInteraction.**get_action_info**(*section*, *action*, *failed_actions*)

Gets all the required Settings for an action. It updates the section with the Settings.

Parameters

- **section** – The section the action corresponds to.
- **action** – The action to get the info for.
- **failed_actions** – A set of all actions that have failed. A failed action remains in the list until it is successfully executed.

Returns Action name and the updated section.

coalib.output.ConsoleInteraction.**highlight_text**(*no_color*, *text*, *lexer=<pygments.lexers.TextLexer>*, *style=None*)

coalib.output.ConsoleInteraction.**join_names**(*values*)

Produces a string by concatenating the items in `values` with commas, except the last element, which is concatenated with an "and".

```
>>> join_names(["apples", "bananas", "oranges"])
'apples, bananas and oranges'
>>> join_names(["apples", "bananas"])
'apples and bananas'
>>> join_names(["apples"])
'apples'
```

Parameters **values** – A list of strings.

Returns The concatenated string.

coalib.output.ConsoleInteraction.**nothing_done**(*log_printer*)
Will be called after processing a coafile when nothing had to be done, i.e. no section was enabled/targeted.

Parameters **log_printer** – A LogPrinter object.

coalib.output.ConsoleInteraction.**print_actions**(*console_printer*, *section*, *actions*, *failed_actions*)
Prints the given actions and lets the user choose.

Parameters

- **console_printer** – Object to print messages on the console.
- **actions** – A list of FunctionMetadata objects.
- **failed_actions** – A set of all actions that have failed. A failed action remains in the list until it is successfully executed.

Returns A tuple with the name member of the FunctionMetadata object chosen by the user and a Section containing at least all needed values for the action. If the user did choose to do nothing, return (None, None).

coalib.output.ConsoleInteraction.**print_affected_files**(*console_printer*, *log_printer*, *result*, *file_dict*)
Prints all the affected files and affected lines within them.

Parameters

- **console_printer** – Object to print messages on the console.
- **log_printer** – Printer responsible for logging the messages.
- **result** – The result to print the context for.
- **file_dict** – A dictionary containing all files with filename as key.

coalib.output.ConsoleInteraction.**print_affected_lines**(*console_printer*, *file_dict*, *sourcerange*)
Prints the lines affected by the bears.

Parameters

- **console_printer** – Object to print messages on the console.
- **file_dict** – A dictionary containing all files with filename as key.
- **sourcerange** – The SourceRange object referring to the related lines to print.

coalib.output.ConsoleInteraction.**print_bears**(*bears*, *show_description*, *show_params*, *console_printer*)
Presents all bears being used in a stylized manner.

Parameters

- **bears** – It's a dictionary with bears as keys and list of sections containing those bears as values.
- **show_description** – True if the main description of the bears should be shown.
- **show_params** – True if the parameters and their description should be shown.
- **console_printer** – Object to print messages on the console.

coalib.output.ConsoleInteraction.**print_diffs_info**(*diffs*, *printer*)

coalib.output.ConsoleInteraction.**print_lines**(*console_printer*, *file_dict*, *sourcerange*)

Prints the lines between the current and the result line. If needed they will be shortened.

Parameters

- **console_printer** – Object to print messages on the console.
- **file_dict** – A dictionary containing all files as values with filenames as key.
- **sourcerange** – The SourceRange object referring to the related lines to print.

coalib.output.ConsoleInteraction.**print_result**(*console_printer*, *section*, *file_diff_dict*, *result*, *file_dict*, *interactive=True*)

Prints the result to console.

Parameters

- **console_printer** – Object to print messages on the console.
- **section** – Name of section to which the result belongs.
- **file_diff_dict** – Dictionary containing filenames as keys and Diff objects as values.
- **result** – A derivative of Result.
- **file_dict** – A dictionary containing all files with filename as key.
- **interactive** – Variable to check wether or not to offer the user actions interactively.

coalib.output.ConsoleInteraction.**print_results**(*log_printer*, *section*, *result_list*, *file_dict*, *file_diff_dict*, *console_printer*)

Prints all the results in a section.

Parameters

- **log_printer** – Printer responsible for logging the messages.
- **section** – The section to which the results belong to.
- **result_list** – List containing the results
- **file_dict** – A dictionary containing all files with filename as key.
- **file_diff_dict** – A dictionary that contains filenames as keys and diff objects as values.
- **console_printer** – Object to print messages on the console.

coalib.output.ConsoleInteraction.**print_results_formatted**(*log_printer*, *section*, *result_list*, **args*)

coalib.output.ConsoleInteraction.**print_results_no_input**(*log_printer*, *section*, *result_list*, *file_dict*, *file_diff_dict*, *console_printer*)

Prints all non interactive results in a section

Parameters

- **log_printer** – Printer responsible for logging the messages.
- **section** – The section to which the results belong to.
- **result_list** – List containing the results
- **file_dict** – A dictionary containing all files with filename as key.
- **file_diff_dict** – A dictionary that contains filenames as keys and diff objects as values.
- **console_printer** – Object to print messages on the console.

coalib.output.ConsoleInteraction.**print_section_beginning**(*console_printer*, *section*)
Will be called after initialization current_section in begin_section()

Parameters

- **console_printer** – Object to print messages on the console.
- **section** – The section that will get executed now.

coalib.output.ConsoleInteraction.**require_setting**(*setting_name*, *arr*, *section*)
This method is responsible for prompting a user about a missing setting and taking its value as input from the user.

Parameters

- **setting_name** – Name of the setting missing
- **arr** – A list containing a description in [0] and the name of the bears who need this setting in [1] and following.
- **section** – The section the action corresponds to.

coalib.output.ConsoleInteraction.**show_bear**(*bear*, *show_description*, *show_params*, *console_printer*)
Displays all information about a bear.

Parameters

- **bear** – The bear to be displayed.
- **show_description** – True if the main description should be shown.
- **show_params** – True if the details should be shown.
- **console_printer** – Object to print messages on the console.

coalib.output.ConsoleInteraction.**show_bears**(*local_bears*, *global_bears*, *show_description*, *show_params*, *console_printer*)
Extracts all the bears from each enabled section or the sections in the targets and passes a dictionary to the show_bears_callback method.

Parameters

- **local_bears** – Dictionary of local bears with section names as keys and bear list as values.
- **global_bears** – Dictionary of global bears with section names as keys and bear list as values.
- **show_description** – True if the main description of the bears should be shown.
- **show_params** – True if the parameters and their description should be shown.
- **console_printer** – Object to print messages on the console.

coalib.output.ConsoleInteraction.**show_enumeration**(*console_printer*, *title*, *items*, *indentation*, *no_items_text*)

This function takes as input an iterable object (preferably a list or a dict) and prints it in a stylized format. If the iterable object is empty, it prints a specific statement given by the user. An e.g :

<indentation>Title: <indentation> * Item 1 <indentation> * Item 2

Parameters

- **console_printer** – Object to print messages on the console.
- **title** – Title of the text to be printed
- **items** – The iterable object.
- **indentation** – Number of spaces to indent every line by.
- **no_items_text** – Text printed when iterable object is empty.

coalib.output.ConsoleInteraction.**show_language_bears_capabilities**(*language_bears_capabilities*, *console_printer*)

Displays what the bears can detect and fix.

Parameters

- **language_bears_capabilities** – Dictionary with languages as keys and their bears' capabilities as values. The capabilities are stored in a tuple of two elements where the first one represents what the bears can detect, and the second one what they can fix.
- **console_printer** – Object to print messages on the console.

coalib.output.Interactions module

coalib.output.Interactions.**fail_acquire_settings**(*log_printer*, *settings_names_dict*, *section*)

This method throws an exception if any setting needs to be acquired.

Parameters

- **log_printer** – Printer responsible for logging the messages.
- **settings** – A dictionary with the settings name as key and a list containing a description in [0] and the name of the bears who need this setting in [1] and following.

Raises

- **AssertionError** – If any setting is required.
- **TypeError** – If `settings_names_dict` is not a dictionary.

coalib.output.JSONEncoder module

coalib.output.JSONEncoder.**create_json_encoder**(***kwargs*)

Module contents

1.1.6 coalib.parsing package

Submodules

coalib.parsing.CliParsing module

coalib.parsing.CliParsing.**check_conflicts**(*sections*)
Checks if there are any conflicting arguments passed.

Parameters **sections** – The `{section_name: section_object}` dictionary to check conflicts for.

Returns True if no conflicts occur.

Raises **SystemExit** – If there are conflicting arguments (exit code: 2)

coalib.parsing.CliParsing.**parse_cli**(*arg_list=None*, *origin='/home/docs/checkouts/readthedocs.org/user_builds/coala-api/checkouts/0.9.0/docs'*, *arg_parser=None*, *key_value_delimiters=('=', ':')*, *comment_seperators=()*, *key_delimiters=(',',)*, *section_override_delimiters=('.',)*)
Parses the CLI arguments and creates sections out of it.

Parameters

- **arg_list** – The CLI argument list.
- **origin** – Directory used to interpret relative paths given as argument.
- **arg_parser** – Instance of ArgParser that is used to parse none-setting arguments.
- **key_value_delimiters** – Delimiters to separate key and value in setting arguments.
- **comment_seperators** – Allowed prefixes for comments.
- **key_delimiters** – Delimiter to separate multiple keys of a setting argument.
- **section_override_delimiters** – The delimiter to delimit the section from the key name (e.g. the '.' in sect.key = value).

Returns A dictionary holding section names as keys and the sections themselves as value.

coalib.parsing.CliParsing.**parse_custom_settings**(*sections*, *custom_settings_list*, *origin*, *line_parser*)
Parses the custom settings given to coala via `-S something=value`.

Parameters

- **sections** – The Section dictionary to add to (mutable).
- **custom_settings_list** – The list of settings strings.
- **origin** – The originating directory.
- **line_parser** – The LineParser to use.

coalib.parsing.ConfParser module

class coalib.parsing.ConfParser.**ConfParser**(*key_value_delimiters=('=',)*, *comment_seperators=('#',)*, *key_delimiters=(',', ' ')*, *section_name_surroundings=mappingproxy({'[': ']'})*, *remove_empty_iter_elements=True*)

Bases: object

get_section(*name*, *create_if_not_exists=False*)

parse(*input_data*, *overwrite=False*)

Parses the input and adds the new data to the existing.

Parameters

- **input_data** – The filename to parse from.
- **overwrite** – If True, wipes all existing Settings inside this instance and adds only the newly parsed ones. If False, adds the newly parsed data to the existing one (and overwrites already existing keys with the newly parsed values).

Returns A dictionary with (lowercase) section names as keys and their Setting objects as values.

coalib.parsing.DefaultArgParser module

class coalib.parsing.DefaultArgParser.**CustomFormatter**(*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*)

Bases: argparse.RawDescriptionHelpFormatter

A Custom Formatter that will keep the metavars in the usage but remove them in the more detailed arguments section.

coalib.parsing.DefaultArgParser.**default_arg_parser**(*formatter_class=None*)

This function creates an ArgParser to parse command line arguments.

Parameters **formatter_class** – Formatting the arg_parser output into a specific form. For example: In the manpage format.

coalib.parsing.Globbing module

coalib.parsing.Globbing.**fnmatch**(*name*, *globs*)

Tests whether name matches one of the given globs.

Parameters

- **name** – File or directory name
- **globs** – Glob string with wildcards or list of globs

Returns Boolean: Whether or not name is matched by glob

Glob Syntax:

- **'[seq]': Matches any character in seq. Cannot be empty. Any** special character looses its special meaning in a set.
- **'[!seq]': Matches any character not in seq. Cannot be empty. Any** special character looses its special meaning in a set.

- **'(seq_a|seq_b)': Matches either sequence_a or sequence_b as a whole.** More than two or just one sequence can be given.
- '?': Matches any single character.
- '*': Matches everything but os.sep.
- '**': Matches everything.

coalib.parsing.Globbing.**glob**(*pattern*)
Iterates all filesystem paths that get matched by the glob pattern. Syntax is equal to that of fnmatch.

Parameters **pattern** – Glob pattern with wildcards

Returns List of all file names that match pattern

coalib.parsing.Globbing.**glob_escape**(*input_string*)
Escapes the given string with `[c]` pattern. Examples:

```
>>> from coalib.parsing.Globbing import glob_escape
>>> glob_escape('test (1)')
'test [(]1[)]'
>>> glob_escape('test folder?')
'test folder[?]'
>>> glob_escape('test*folder')
'test[*]folder'
```

Parameters **input_string** – String that is to be escaped with `[ ]`.

Returns Escaped string in which all the special glob characters `()[]|?*` are escaped.

coalib.parsing.Globbing.**has_wildcard**(*pattern*)
Checks whether pattern has any wildcards.

Parameters **pattern** – Glob pattern that may contain wildcards

Returns Boolean: Whether or not there are wildcards in pattern

coalib.parsing.Globbing.**iglob**(*pattern*)
Iterates all filesystem paths that get matched by the glob pattern. Syntax is equal to that of fnmatch.

Parameters **pattern** – Glob pattern with wildcards

Returns Iterator that yields all file names that match pattern

coalib.parsing.Globbing.**relative_flat_glob**(*dirname*, *basename*)
Non-recursive glob for one directory. Does not accept wildcards.

Parameters

- **dirname** – Directory name
- **basename** – Basename of a file in dir of dirname

Returns List containing Basename if the file exists

coalib.parsing.Globbing.**relative_recursive_glob**(*dirname*, *pattern*)
Recursive Glob for one directory and all its (nested) subdirectories. Accepts only '**' as pattern.

Parameters

- **dirname** – Directory name
- **pattern** – The recursive wildcard '**'

Returns Iterator that yields all the (nested) subdirectories of the given dir

coalib.parsing.Globbing.**relative_wildcard_glob**(*dirname*, *pattern*)
Non-recursive glob for one directory. Accepts wildcards.

Parameters

- **dirname** – Directory name
- **pattern** – Glob pattern with wildcards

Returns List of files in the dir of dirname that match the pattern

coalib.parsing.Globbing.**translate**(*pattern*)
Translates a pattern into a regular expression.

Parameters **pattern** – Glob pattern with wildcards

Returns Regular expression with the same meaning

coalib.parsing.LineParser module

class coalib.parsing.LineParser.**LineParser**(*key_value_delimiters=('=',)*, *comment_separators=('#',)*, *key_delimiters=(',', ' ')*, *section_name_surroundings=None*, *section_override_delimiters=('.',)*)
Bases: object

parse(*line*)
Note that every value in the returned tuple *besides the value* is unescaped. This is so since the value is meant to be put into a Setting later thus the escapes may be needed there.

Parameters **line** – The line to parse.

Returns section_name (empty string if it's no section name), [(section_override, key), ...], value, comment

Module contents

The StringProcessing module contains various functions for extracting information out of strings.

Most of them support regexes for advanced pattern matching.

1.1.7 coalib.processes package

Subpackages

coalib.processes.communication package

Submodules

coalib.processes.communication.LogMessage module

class coalib.processes.communication.LogMessage.**LogMessage**(*log_level*, **messages*, ***, *delimiter=' '*, *timestamp=None*)
Bases: object

to_string_dict()
Makes a dictionary which has all keys and values as strings and contains all the data that the LogMessage has.

Returns Dictionary with keys and values as string.

Module contents

Submodules

coalib.processes.BearRunning module

coalib.processes.BearRunning.**get_global_dependency_results**(*global_result_dict*, *bear_instance*)
This method gets all the results originating from the dependencies of a bear_instance. Each bear_instance may or may not have dependencies.

Parameters **global_result_dict** – The list of results out of which the dependency results are picked.

Returns None if bear has no dependencies, False if dependencies are not met, the dependency dict otherwise.

coalib.processes.BearRunning.**get_local_dependency_results**(*local_result_list*, *bear_instance*)
This method gets all the results originating from the dependencies of a bear_instance. Each bear_instance may or may not have dependencies.

Parameters

- **local_result_list** – The list of results out of which the dependency results are picked.
- **bear_instance** – The instance of a local bear to get the dependencies from.

Returns Return none if there are no dependencies for the bear. Else return a dictionary containing dependency results.

coalib.processes.BearRunning.**get_next_global_bear**(*timeout*, *global_bear_queue*, *global_bear_list*, *global_result_dict*)
Retrieves the next global bear.

Parameters

- **timeout** – The queue blocks at most timeout seconds for a free slot to execute the put operation on. After the timeout it returns queue Full exception.
- **global_bear_queue** – queue (read, write) of indexes of global bear instances in the global_bear_list.
- **global_bear_list** – A list containing all global bears to be executed.
- **global_result_dict** – A Manager.dict that will be used to store global results. The list of results of one global bear will be stored with the bear name as key.

Returns (bear, bearname, dependency_results)

coalib.processes.BearRunning.**run**(*file_name_queue*, *local_bear_list*, *global_bear_list*, *global_bear_queue*, *file_dict*, *local_result_dict*, *global_result_dict*, *message_queue*, *control_queue*, *timeout=0*)

This is the method that is actually runs by processes.

If parameters type is 'queue (read)' this means it has to implement the get(timeout=TIMEOUT) method and it shall raise queue.Empty if the queue is empty up until the end of the timeout. If the queue has the (optional!) task_done() attribute, the run method will call it after processing each item.

If parameters type is 'queue (write)' it shall implement the put(object, timeout=TIMEOUT) method.

If the queues raise any exception not specified here the user will get an 'unknown error' message. So beware of that.

Parameters

- **file_name_queue** – queue (read) of file names to check with local bears. Each invocation of the run method needs one such queue which it checks with all the local bears. The queue could be empty. (Repeat until queue empty.)
- **local_bear_list** – List of local bear instances.
- **global_bear_list** – List of global bear instances.
- **global_bear_queue** – queue (read, write) of indexes of global bear instances in the global_bear_list.
- **file_dict** – dict of all files as {filename:file}, file as in file.readlines().
- **local_result_dict** – A Manager.dict that will be used to store local results. A list of all local results. will be stored with the filename as key.
- **global_result_dict** – A Manager.dict that will be used to store global results. The list of results of one global bear will be stored with the bear name as key.
- **message_queue** – queue (write) for debug/warning/error messages (type LogMessage)
- **control_queue** – queue (write). If any result gets written to the result_dict a tuple containing a CONTROL_ELEMENT (to indicate what kind of event happened) and either a bear name (for global results) or a file name to indicate the result will be put to the queue. If the run method finished all its local bears it will put (CONTROL_ELEMENT.LOCAL_FINISHED, None) to the queue, if it finished all global ones, (CONTROL_ELEMENT.GLOBAL_FINISHED, None) will be put there.
- **timeout** – The queue blocks at most timeout seconds for a free slot to execute the put operation on. After the timeout it returns queue Full exception.

coalib.processes.BearRunning.**run_bear**(*message_queue*, *timeout*, *bear_instance*, **args*, ***kwargs*)

This method is responsible for executing the instance of a bear. It also reports or logs errors if any occur during the execution of that bear instance.

Parameters

- **message_queue** – A queue that contains messages of type errors/warnings/debug statements to be printed in the Log.
- **timeout** – The queue blocks at most timeout seconds for a free slot to execute the put operation on. After the timeout it returns queue Full exception.
- **bear_instance** – The instance of the bear to be executed.
- **args** – The arguments that are to be passed to the bear.

- **kwargs** – The keyword arguments that are to be passed to the bear.

Returns Returns a valid list of objects of the type Result if the bear executed successfully. None otherwise.

coalib.processes.BearRunning.**run_global_bear**(*message_queue*, *timeout*, *global_bear_instance*, *dependency_results*)

Runs an instance of a global bear. Checks if bear_instance is of type GlobalBear and then passes it to the run_bear to execute.

Parameters

- **message_queue** – A queue that contains messages of type errors/warnings/debug statements to be printed in the Log.
- **timeout** – The queue blocks at most timeout seconds for a free slot to execute the put operation on. After the timeout it returns queue Full exception.
- **global_bear_instance** – Instance of GlobalBear to run.
- **dependency_results** – The results of all the bears on which the instance of the passed bear to be run depends on.

Returns Returns a list of results generated by the passed bear_instance.

coalib.processes.BearRunning.**run_global_bears**(*message_queue*, *timeout*, *global_bear_queue*, *global_bear_list*, *global_result_dict*, *control_queue*)

Run all global bears.

Parameters

- **message_queue** – A queue that contains messages of type errors/warnings/debug statements to be printed in the Log.
- **timeout** – The queue blocks at most timeout seconds for a free slot to execute the put operation on. After the timeout it returns queue Full exception.
- **global_bear_queue** – queue (read, write) of indexes of global bear instances in the global_bear_list.
- **global_bear_list** – list of global bear instances
- **global_result_dict** – A Manager.dict that will be used to store global results. The list of results of one global bear will be stored with the bear name as key.
- **control_queue** – If any result gets written to the result_dict a tuple containing a CONTROL_ELEMENT (to indicate what kind of event happened) and either a bear name(for global results) or a file name to indicate the result will be put to the queue.

coalib.processes.BearRunning.**run_local_bear**(*message_queue*, *timeout*, *local_result_list*, *file_dict*, *bear_instance*, *filename*)

Runs an instance of a local bear. Checks if bear_instance is of type LocalBear and then passes it to the run_bear to execute.

Parameters

- **message_queue** – A queue that contains messages of type errors/warnings/debug statements to be printed in the Log.
- **timeout** – The queue blocks at most timeout seconds for a free slot to execute the put operation on. After the timeout it returns queue Full exception.
- **local_result_list** – Its a list that stores the results of all local bears.

- **file_dict** – Dictionary containing contents of file.
- **bear_instance** – Instance of LocalBear the run.
- **filename** – Name of the file to run it on.

Returns Returns a list of results generated by the passed bear_instance.

coalib.processes.BearRunning.**run_local_bears**(*filename_queue*, *message_queue*, *timeout*, *file_dict*, *local_bear_list*, *local_result_dict*, *control_queue*)

Run local bears on all the files given.

Parameters

- **filename_queue** – queue (read) of file names to check with local bears.
- **message_queue** – A queue that contains messages of type errors/warnings/debug statements to be printed in the Log.
- **timeout** – The queue blocks at most timeout seconds for a free slot to execute the put operation on. After the timeout it returns queue Full exception.
- **file_dict** – Dictionary that contains contents of files.
- **local_bear_list** – List of local bears to run.
- **local_result_dict** – A Manager.dict that will be used to store local bear results. A list of all local bear results will be stored with the filename as key.
- **control_queue** – If any result gets written to the result_dict a tuple containing a CONTROL_ELEMENT (to indicate what kind of event happened) and either a bear name(for global results) or a file name to indicate the result will be put to the queue.

coalib.processes.BearRunning.**run_local_bears_on_file**(*message_queue*, *timeout*, *file_dict*, *local_bear_list*, *local_result_dict*, *control_queue*, *filename*)

This method runs a list of local bears on one file.

Parameters

- **message_queue** – A queue that contains messages of type errors/warnings/debug statements to be printed in the Log.
- **timeout** – The queue blocks at most timeout seconds for a free slot to execute the put operation on. After the timeout it returns queue Full exception.
- **file_dict** – Dictionary that contains contents of files.
- **local_bear_list** – List of local bears to run on file.
- **local_result_dict** – A Manager.dict that will be used to store local bear results. A list of all local bear results will be stored with the filename as key.
- **control_queue** – If any result gets written to the result_dict a tuple containing a CONTROL_ELEMENT (to indicate what kind of event happened) and either a bear name(for global results) or a file name to indicate the result will be put to the queue.
- **filename** – The name of file on which to run the bears.

coalib.processes.BearRunning.**send_msg**(*message_queue*, *timeout*, *log_level*, **args*, ***, *delimiter=' '*, *end=''*)

Puts message into message queue for a LogPrinter to present to the user.

Parameters

- **message_queue** – The queue to put the message into and which the LogPrinter reads.
- **timeout** – The queue blocks at most timeout seconds for a free slot to execute the put operation on. After the timeout it returns queue Full exception.
- **log_level** – The log_level i.e Error,Debug or Warning.It is sent to the LogPrinter depending on the message.
- **args** – This includes the elements of the message.
- **delimiter** – It is the value placed between each arg. By default it is a ' '.
- **end** – It is the value placed at the end of the message.

coalib.processes.BearRunning.**task_done**(*obj*)

Invokes task_done if the given queue provides this operation. Otherwise passes silently.

Parameters **obj** – Any object.

coalib.processes.BearRunning.**validate_results**(*message_queue*, *timeout*, *result_list*, *name*, *args*, *kwargs*)

Validates if the result_list passed to it contains valid set of results. That is the result_list must itself be a list and contain objects of the instance of Result object. If any irregularity is found a message is put in the message_queue to present the irregularity to the user. Each result_list belongs to an execution of a bear.

Parameters

- **message_queue** – A queue that contains messages of type errors/warnings/debug statements to be printed in the Log.
- **timeout** – The queue blocks at most timeout seconds for a free slot to execute the put operation on. After the timeout it returns queue Full exception.
- **result_list** – The list of results to validate.
- **name** – The name of the bear executed.
- **args** – The args with which the bear was executed.
- **kwargs** – The kwargs with which the bear was executed.

Returns Returns None if the result_list is invalid. Else it returns the result_list itself.

coalib.processes.CONTROL_ELEMENT module

coalib.processes.LogPrinterThread module

class coalib.processes.LogPrinterThread.**LogPrinterThread**(*message_queue*, *log_printer*)

Bases: threading.Thread

This is the Thread object that outputs all log messages it gets from its message_queue. Setting obj.running = False will stop within the next 0.1 seconds.

run()

coalib.processes.Processing module

coalib.processes.Processing.**autoapply_actions**(*results*, *file_dict*, *file_diff_dict*, *section*, *log_printer*)

Auto-applies actions like defined in the given section.

Parameters

- **results** – A list of results.
- **file_dict** – A dictionary containing the name of files and its contents.
- **file_diff_dict** – A dictionary that contains filenames as keys and diff objects as values.
- **section** – The section.
- **log_printer** – A log printer instance to log messages on.

Returns A list of unprocessed results.

coalib.processes.Processing.**check_result_ignore**(*result*, *ignore_ranges*)
Determines if the result has to be ignored.

Parameters

- **result** – The result that needs to be checked.
- **ignore_ranges** – A list of tuples, each containing a list of lower cased affected bearnames and a SourceRange to ignore. If any of the bearname lists is empty, it is considered an ignore range for all bears. This may be a list of globbed bear wildcards.

Returns True if the result has to be ignored.

coalib.processes.Processing.**create_process_group**(*command_array*, ***kwargs*)

coalib.processes.Processing.**execute_section**(*section*, *global_bear_list*, *local_bear_list*, *print_results*, *cache*, *log_printer*, *console_printer*)
Executes the section with the given bears.

The execute_section method does the following things:

1.Prepare a Process - Load files - Create queues

2.Spawn up one or more Processes

3.Output results from the Processes

4.Join all processes

Parameters

- **section** – The section to execute.
- **global_bear_list** – List of global bears belonging to the section. Dependencies are already resolved.
- **local_bear_list** – List of local bears belonging to the section. Dependencies are already resolved.
- **print_results** – Prints all given results appropriate to the output medium.
- **cache** – An instance of misc.Caching.FileCache to use as a file cache buffer.
- **log_printer** – The log_printer to warn to.
- **console_printer** – Object to print messages on the console.

Returns Tuple containing a bool (True if results were yielded, False otherwise), a Manager.dict containing all local results(filenames are key) and a Manager.dict containing all global bear results (bear names are key) as well as the file dictionary.

coalib.processes.Processing.**fill_queue**(*queue_fill*, *any_list*)
Takes element from a list and populates a queue with those elements.

Parameters

- **queue_fill** – The queue to be filled.
- **any_list** – List containing the elements.

coalib.processes.Processing.**filter_raising_callables**(*it*, *exception*, **args*, ***kwargs*)
Filters all callable items inside the given iterator that raise the given exceptions.

Parameters

- **it** – The iterator to filter.
- **exception** – The (tuple of) exception(s) to filter for.
- **args** – Positional arguments to pass to the callable.
- **kwargs** – Keyword arguments to pass to the callable.

coalib.processes.Processing.**get_cpu_count**()

coalib.processes.Processing.**get_default_actions**(*section*)
Parses the key `default_actions` in the given section.

Parameters **section** – The section where to parse from.

Returns A dict with the bearname as keys and their default actions as values and another dict that contains bears and invalid action names.

coalib.processes.Processing.**get_file_dict**(*filename_list*, *log_printer*)
Reads all files into a dictionary.

Parameters

- **filename_list** – List of names of paths to files to get contents of.
- **log_printer** – The logger which logs errors.

Returns Reads the content of each file into a dictionary with filenames as keys.

coalib.processes.Processing.**get_file_list**(*results*)
Get the set of files that are affected in the given results.

Parameters **results** – A list of results from which the list of files is to be extracted.

Returns A set of file paths containing the mentioned list of files.

coalib.processes.Processing.**get_ignore_scope**(*line*, *keyword*)
Retrieves the bears that are to be ignored defined in the given line.

Parameters

- **line** – The line containing the ignore declaration.
- **keyword** – The keyword that was found. Everything after the rightmost occurrence of it will be considered for the scope.

Returns A list of lower cased bearnames or an empty list (-> “all”)

coalib.processes.Processing.**get_running_processes**(*processes*)

coalib.processes.Processing.**instantiate_bears**(*section*, *local_bear_list*, *global_bear_list*, *file_dict*, *message_queue*, *console_printer*)
Instantiates each bear with the arguments it needs.

Parameters

- **section** – The section the bears belong to.

- **local_bear_list** – List of local bear classes to instantiate.
- **global_bear_list** – List of global bear classes to instantiate.
- **file_dict** – Dictionary containing filenames and their contents.
- **message_queue** – Queue responsible to maintain the messages delivered by the bears.
- **console_printer** – Object to print messages on the console.

Returns The local and global bear instance lists.

coalib.processes.Processing.**instantiate_processes**(*section*, *local_bear_list*, *global_bear_list*, *job_count*, *cache*, *log_printer*, *console_printer*)

Instantiate the number of processes that will run bears which will be responsible for running bears in a multiprocessing environment.

Parameters

- **section** – The section the bears belong to.
- **local_bear_list** – List of local bears belonging to the section.
- **global_bear_list** – List of global bears belonging to the section.
- **job_count** – Max number of processes to create.
- **cache** – An instance of `misc.Caching.FileCache` to use as a file cache buffer.
- **log_printer** – The log printer to warn to.
- **console_printer** – Object to print messages on the console.

Returns A tuple containing a list of processes, and the arguments passed to each process which are the same for each object.

coalib.processes.Processing.**print_result**(*results*, *file_dict*, *retval*, *print_results*, *section*, *log_printer*, *file_diff_dict*, *ignore_ranges*, *console_printer*)

Takes the results produced by each bear and gives them to the print_results method to present to the user.

Parameters

- **results** – A list of results.
- **file_dict** – A dictionary containing the name of files and its contents.
- **retval** – It is True if no results were yielded ever before. If it is False this function will return False no matter what happens. Else it depends on if this invocation yields results.
- **print_results** – A function that prints all given results appropriate to the output medium.
- **file_diff_dict** – A dictionary that contains filenames as keys and diff objects as values.
- **ignore_ranges** – A list of SourceRanges. Results that affect code in any of those ranges will be ignored.
- **console_printer** – Object to print messages on the console.

Returns Returns False if any results were yielded. Else True.

coalib.processes.Processing.**process_queues**(*processes*, *control_queue*, *local_result_dict*, *global_result_dict*, *file_dict*, *print_results*, *section*, *cache*, *log_printer*, *console_printer*)

Iterate the control queue and send the results received to the print_result method so that they can be presented to the user.

Parameters

- **processes** – List of processes which can be used to run Bears.
- **control_queue** – Containing control elements that indicate whether there is a result available and which bear it belongs to.
- **local_result_dict** – Dictionary containing results respective to local bears. It is modified by the processes i.e. results are added to it by multiple processes.
- **global_result_dict** – Dictionary containing results respective to global bears. It is modified by the processes i.e. results are added to it by multiple processes.
- **file_dict** – Dictionary containing file contents with filename as keys.
- **print_results** – Prints all given results appropriate to the output medium.
- **cache** – An instance of `misc.Caching.FileCache` to use as a file cache buffer.

Returns Return True if all bears execute successfully and Results were delivered to the user. Else False.

coalib.processes.Processing.**simplify_section_result**(*section_result*)

Takes in a section's result from `execute_section` and simplifies it for easy usage in other functions.

Parameters **section_result** – The result of a section which was executed.

Returns Tuple containing: - bool - True if results were yielded - bool - True if unfixed results were yielded - list - Results from all bears (local and global)

coalib.processes.Processing.**yield_ignore_ranges**(*file_dict*)

Yields tuples of affected bears and a SourceRange that shall be ignored for those.

Parameters **file_dict** – The file dictionary.

Module contents

1.1.8 coalib.results package

Subpackages

coalib.results.result_actions package

Submodules

coalib.results.result_actions.ApplyPatchAction module

class coalib.results.result_actions.ApplyPatchAction.**ApplyPatchAction**

Bases: *coalib.results.result_actions.ResultAction.ResultAction*

SUCCESS_MESSAGE = 'Patch applied successfully.'

apply(*result*, *original_file_dict*, *file_diff_dict*, *no_orig: bool = False*)

Apply patch

Parameters **no_orig** – Whether or not to create .orig backup files

static is_applicable(*result*, *original_file_dict*, *file_diff_dict*)

coalib.results.result_actions.IgnoreResultAction module

class coalib.results.result_actions.IgnoreResultAction.**IgnoreResultAction**
Bases: *coalib.results.result_actions.ResultAction.ResultAction*

SUCCESS_MESSAGE = 'An ignore comment was added to your source code.'

apply(*result*, *original_file_dict*, *file_diff_dict*, *language: str*, *no_orig: bool = False*)
Add ignore comment

static is_applicable(*result*, *original_file_dict*, *file_diff_dict*)
For being applicable, the result has to point to a number of files that have to exist i.e. have not been previously deleted.

coalib.results.result_actions.OpenEditorAction module

class coalib.results.result_actions.OpenEditorAction.**OpenEditorAction**
Bases: *coalib.results.result_actions.ResultAction.ResultAction*

SUCCESS_MESSAGE = 'Changes saved successfully.'

apply(*result*, *original_file_dict*, *file_diff_dict*, *editor: str*)
Open file(s)

Parameters **editor** – The editor to open the file with.

static is_applicable(*result*, *original_file_dict*, *file_diff_dict*)
For being applicable, the result has to point to a number of files that have to exist i.e. have not been previously deleted.

coalib.results.result_actions.PrintDebugMessageAction module

class coalib.results.result_actions.PrintDebugMessageAction.**PrintDebugMessageAction**
Bases: *coalib.results.result_actions.ResultAction.ResultAction*

apply(*result*, *original_file_dict*, *file_diff_dict*)
Print debug message

static is_applicable(*result*, *original_file_dict*, *file_diff_dict*)

coalib.results.result_actions.PrintMoreInfoAction module

class coalib.results.result_actions.PrintMoreInfoAction.**PrintMoreInfoAction**
Bases: *coalib.results.result_actions.ResultAction.ResultAction*

apply(*result*, *original_file_dict*, *file_diff_dict*)
Print more info

static is_applicable(*result*, *original_file_dict*, *file_diff_dict*)

coalib.results.result_actions.ResultAction module

A ResultAction is an action that is applicable to at least some results. This file serves the base class for all result actions, thus providing a unified interface for all actions.

class coalib.results.result_actions.ResultAction.**ResultAction**
Bases: object

SUCCESS_MESSAGE = 'The action was executed successfully.'

apply(*result*, *original_file_dict*, *file_diff_dict*, ***kwargs*)
No description. Something went wrong.

apply_from_section(*result*, *original_file_dict: dict*, *file_diff_dict: dict*, *section: coalib.settings.Section.Section*)
Applies this action to the given results with all additional options given as a section. The file dictionaries are needed for differential results.

Parameters

- **result** – The result to apply.
- **original_file_dict** – A dictionary containing the files in the state where the result was generated.
- **file_diff_dict** – A dictionary containing a diff for every file from the state in the original_file_dict to the current state. This dict will be altered so you do not need to use the return value.
- **section** – The section where to retrieve the additional information.

Returns The modified file_diff_dict.

classmethod get_metadata()
Retrieves metadata for the apply function. The description may be used to advertise this action to the user. The parameters and their help texts are additional information that are needed from the user. You can create a section out of the inputs from the user and use apply_from_section to apply

:return A FunctionMetadata object.

static is_applicable(*result*, *original_file_dict*, *file_diff_dict*)
Checks whether the Action is valid for the result type.

Returns True by default.

Parameters

- **result** – The result from the coala run to check if an Action is applicable.
- **original_file_dict** – A dictionary containing the files in the state where the result was generated.
- **file_diff_dict** – A dictionary containing a diff for every file from the state in the original_file_dict to the current state. This dict will be altered so you do not need to use the return value.

coalib.results.result_actions.ShowPatchAction module

class coalib.results.result_actions.ShowPatchAction.**ShowPatchAction**
Bases: *coalib.results.result_actions.ResultAction.ResultAction*

SUCCESS_MESSAGE = 'Displayed patch successfully.'

apply (*result*, *original_file_dict*, *file_diff_dict*, *colored: bool = True*, *show_result_on_top: bool = False*)
Show patch

Parameters

- **colored** – Whether or not to use colored output.
- **show_result_on_top** – Set this to True if you want to show the result info on top. (Useful for e.g. coala_ci.)

static is_applicable (*result*, *original_file_dict*, *file_diff_dict*)

coalib.results.result_actions.ShowPatchAction.**format_line** (*line*, *real_nr=''*, *sign='|'*, *mod_nr=''*, *symbol=''*)

coalib.results.result_actions.ShowPatchAction.**print_beautified_diff** (*difflines*, *printer*)

coalib.results.result_actions.ShowPatchAction.**print_from_name** (*printer*, *line*)

coalib.results.result_actions.ShowPatchAction.**print_to_name** (*printer*, *line*)

Module contents

The result_actions package holds objects deriving from ResultAction. A ResultAction represents an action that an be applied to a result.

Submodules

coalib.results.AbsolutePosition module

class coalib.results.AbsolutePosition.**AbsolutePosition** (*text: (<class 'tuple'>, <class 'list'>, None) = None*, *position: (<class 'int'>, None) = None*)

Bases: *coalib.results.TextPosition.TextPosition*

position

coalib.results.AbsolutePosition.**calc_line_col** (*text*, *position*)
Creates a tuple containing (line, column) by calculating line number and column in the text, from position.

The position represents the index of a character. In the following example 'a' is at position '0' and it's corresponding line and column are:

```
>>> calc_line_col(('a\n',), 0)
(1, 1)
```

All special characters(including the newline character) belong in the same line, and have their own position. A line is an item in the tuple:

```
>>> calc_line_col(('a\n', 'b\n'), 1)
(1, 2)
>>> calc_line_col(('a\n', 'b\n'), 2)
(2, 1)
```

Parameters

- **text** – A tuple/list of lines in which position is to be calculated.

- **position** – Position (starting from 0) of character to be found in the (line, column) form.

Returns A tuple of the form (line, column), where both line and column start from 1.

coalib.results.Diff module

class coalib.results.Diff.**Diff**(*file_list*, *rename=False*, *delete=False*)

Bases: object

A Diff result represents a difference for one file.

add_lines(*line_nr_before*, *lines*)

Adds lines after the given line number.

Parameters

- **line_nr_before** – Line number of the line before the additions. Use 0 for insert lines before everything.
- **lines** – A list of lines to add.

affected_code(*filename*)

Creates a list of SourceRange objects which point to the related code. Changes on continuous lines will be put into one SourceRange.

Parameters **filename** – The filename to associate the SourceRange's to.

Returns A list of all related SourceRange objects.

change_line(*line_nr*, *original_line*, *replacement*)

Changes the given line with the given line number. The replacement will be there instead.

Given an empty diff object:

```
>>> diff = Diff(['Hey there! Gorgeous.\n',
...              "It's nice that we're here.\n"])
```

We can change a line easily:

```
>>> diff.change_line(1,
...                  'Hey there! Gorgeous.\n',
...                  'Hey there! This is sad.\n')
>>> diff.modified
['Hey there! This is sad.\n', "It's nice that we're here.\n"]
```

We can even merge changes within one line:

```
>>> diff.change_line(1,
...                  'Hey there! Gorgeous.\n',
...                  'Hello. :( Gorgeous.\n')
>>> diff.modified
['Hello. :( This is sad.\n', "It's nice that we're here.\n"]
```

However, if we change something that has been changed before, we'll get a conflict:

```
>>> diff.change_line(1,  # +ELLIPSIS
...                  'Hey there! Gorgeous.\n',
...                  'Hello. This is not ok. Gorgeous.\n')
Traceback (most recent call last):
 ...
coalib.results.LineDiff.ConflictError: ...
```

delete

Returns True if file is set to be deleted.

delete_line (*line_nr*)
Mark the given line nr as deleted. The first line is line number 1.

delete_lines (*line_nr_start*, *line_nr_end*)
Delete lines in a specified range, inclusively.

classmethod from_clang_fixit (*fixit*, *file*)
Creates a Diff object from a given clang fixit and the file contents.

Parameters

- **fixit** – A cindex.Fixit object.
- **file** – A list of lines in the file to apply the fixit to.

Returns The corresponding Diff object.

classmethod from_string_arrays (*file_array_1*, *file_array_2*, *rename=False*)
Creates a Diff object from two arrays containing strings.

If this Diff is applied to the original array, the second array will be created.

Parameters

- **file_array_1** – Original array
- **file_array_2** – Array to compare
- **rename** – False or str containing new name of file.

insert (*position*, *text*)
Inserts (multiline) text at arbitrary position.

```
>>> from coalib.results.TextPosition import TextPosition
>>> test_text = ['123\n', '456\n', '789\n']
>>> def insert(position, text):
...     diff = Diff(test_text)
...     diff.insert(position, text)
...     return diff.modified
>>> insert(TextPosition(2, 3), 'woopy doopy')
['123\n', '45woopy doopy6\n', '789\n']
>>> insert(TextPosition(1, 1), 'woopy\ndoopy')
['woopy\n', 'doopy123\n', '456\n', '789\n']
>>> insert(TextPosition(2, 4), '\nwoopy\ndoopy\n')
['123\n', '456\n', 'woopy\n', 'doopy\n', '\n', '789\n']
```

Parameters

- **position** – The `TextPosition` where to insert text.
- **text** – The text to insert.

modified
Calculates the modified file, after applying the Diff to the original.

original
Retrieves the original file.

range (*filename*)
Calculates a SourceRange spanning over the whole Diff. If something is added after the 0th line (i.e. before the first line) the first line will be included in the SourceRange.

The range of an empty diff will only affect the filename:

```
>>> range = Diff([]).range("file")
>>> range.file is None
False
>>> print(range.start.line)
None
```

Parameters **filename** – The filename to associate the SourceRange with.

Returns A SourceRange object.

remove (*range*)
Removes a piece of text in a given range.

```
>>> from coalib.results.TextRange import TextRange
>>> test_text = ['nice\n', 'try\n', 'bro\n']
>>> def remove(range):
...     diff = Diff(test_text)
...     diff.remove(range)
...     return diff.modified
>>> remove(TextRange.from_values(1, 1, 1, 4))
['e\n', 'try\n', 'bro\n']
>>> remove(TextRange.from_values(1, 5, 2, 1))
['nicetry\n', 'bro\n']
>>> remove(TextRange.from_values(1, 3, 3, 2))
['niro\n']
>>> remove(TextRange.from_values(2, 1, 2, 1))
['nice\n', 'try\n', 'bro\n']
```

Parameters **range** – The range to delete.

rename

Returns string containing new name of the file.

replace (*range*, *replacement*)
Replaces a part of text. Allows to span multiple lines.

This function uses `add_lines` and `delete_lines` accordingly, so calls of those functions on lines given `range` affects after usage or vice versa lead to `ConflictError`.

```
>>> from coalib.results.TextRange import TextRange
>>> test_text = ['hello\n', 'world\n', '4lines\n', 'done\n']
>>> def replace(range, text):
...     diff = Diff(test_text)
...     diff.replace(range, text)
...     return diff.modified
>>> replace(TextRange.from_values(1, 5, 4, 3), '\nyeah\ncool\nno')
['hell\n', 'yeah\n', 'cool\n', 'none\n']
>>> replace(TextRange.from_values(2, 1, 3, 5), 'b')
['hello\n', 'bes\n', 'done\n']
>>> replace(TextRange.from_values(1, 6, 4, 3), '')
['hellone\n']
```

Parameters

- **range** – The `TextRange` that gets replaced.
- **replacement** – The replacement string. Can be multiline.

split_diff(*distance=1*)

Splits this diff into small pieces, such that several continuously altered lines are still together in one diff. All subdiffs will be yielded.

A diff like this with changes being together closely won't be splitted:

```
>>> diff = Diff.from_string_arrays([     'b', 'c', 'e'],
...                                ['a', 'b', 'd', 'f'])
>>> len(list(diff.split_diff()))
1
```

If we set the distance to 0, it will be splitted:

```
>>> len(list(diff.split_diff(distance=0)))
2
```

If a negative distance is given, every change will be yielded as an own diff, even if they are right beneath each other:

```
>>> len(list(diff.split_diff(distance=-1)))
3
```

If a file gets renamed or deleted only, it will be yielded as is:

```
>>> len(list(Diff([], rename='test').split_diff()))
1
```

An empty diff will not yield any diffs:

```
>>> len(list(Diff([]).split_diff()))
0
```

Parameters **distance** – Number of unchanged lines that are allowed in between two changed lines so they get yielded as one diff.

stats()

Returns tuple containing number of additions and deletions in the diff.

unified_diff

Generates a unified diff corresponding to this patch.

Note that the unified diff is not deterministic and thus not suitable for equality comparison.

coalib.results.HiddenResult module

class coalib.results.HiddenResult.**HiddenResult**(*origin*, *contents*)

Bases: *coalib.results.Result.Result*

This is a result that is not meant to be shown to the user. It can be used to transfer any data from a dependent bear to others.

coalib.results.LineDiff module

exception coalib.results.LineDiff.**ConflictError**
Bases: Exception

class coalib.results.LineDiff.**LineDiff**(*change=False*, *delete=False*, *add_after=False*)
Bases: object

A LineDiff holds the difference between two strings.

add_after

change

delete

coalib.results.RESULT_SEVERITY module

coalib.results.Result module

class coalib.results.Result.**Result**(*origin*, *message: str*, *affected_code: (<class 'tuple'>, <class 'list'>) = ()*, *severity: int = 1*, *additional_info: str = ''*, *debug_msg=''*, *diffs: (<class 'dict'>, None) = None*, *confidence: int = 100*, *aspect: (<class 'coalib.bearlib.aspects.base.aspectbase'>, None) = None*)
Bases: object

A result is anything that has an origin and a message.

Optionally it might affect a file.

apply(*file_dict: dict*)
Applies all contained diffs to the given file_dict. This operation will be done in-place.

Parameters **file_dict** – A dictionary containing all files with filename as key and all lines a value. Will be modified.

classmethod **from_values**(*origin*, *message: str*, *file: str*, *line: (<class 'int'>, None) = None*, *column: (<class 'int'>, None) = None*, *end_line: (<class 'int'>, None) = None*, *end_column: (<class 'int'>, None) = None*, *severity: int = 1*, *additional_info: str = ''*, *debug_msg=''*, *diffs: (<class 'dict'>, None) = None*, *confidence: int = 100*, *aspect: (<class 'coalib.bearlib.aspects.base.aspectbase'>, None) = None*)
Creates a result with only one SourceRange with the given start and end locations.

Parameters

- **origin** – Class name or creator object of this object.
- **message** – Message to show with this result.
- **file** – The related file.
- **line** – The first related line in the file. (First line is 1)
- **column** – The column indicating the first character. (First character is 1)
- **end_line** – The last related line in the file.
- **end_column** – The column indicating the last character.
- **severity** – Severity of this result.

- **additional_info** – A long description holding additional information about the issue and/or how to fix it. You can use this like a manual entry for a category of issues.
- **debug_msg** – A message which may help the user find out why this result was yielded.
- **diffs** – A dictionary with filenames as key and a sequence of `Diff` objects associated with them as values.
- **confidence** – A number between 0 and 100 describing the likelihood of this result being a real issue.
- **aspect** – An Aspect object which this result is associated to. Note that this should be a leaf of the aspect tree! (If you have a node, spend some time figuring out which of the leafs exactly your result belongs to.)

location_repr()
Retrieves a string, that briefly represents the affected code of the result.

Returns A string containing all of the affected files separated by a comma.

overlaps(*ranges*)
Determines if the result overlaps with source ranges provided.

Parameters **ranges** – A list SourceRange objects to check for overlap.

Returns True if the ranges overlap with the result.

to_string_dict()
Makes a dictionary which has all keys and values as strings and contains all the data that the base Result has.

FIXME: diffs are not serialized ATM. FIXME: Only the first SourceRange of affected_code is serialized. If there are more, this data is currently missing.

Returns Dictionary with keys and values as string.

coalib.results.ResultFilter module

coalib.results.ResultFilter.**basics_match**(*original_result*, *modified_result*)
Checks whether the following properties of two results match: * origin * message * severity * debug_msg

Parameters

- **original_result** – A result of the old files
- **modified_result** – A result of the new files

Returns Boolean value whether or not the properties match

coalib.results.ResultFilter.**ensure_files_present**(*original_file_dict*, *modified_file_dict*)
Ensures that all files are available as keys in both dicts.

Parameters

- **original_file_dict** – Dict of lists of file contents before changes
- **modified_file_dict** – Dict of lists of file contents after changes

Returns Return a dictionary of renamed files.

coalib.results.ResultFilter.**filter_results**(*original_file_dict*, *modified_file_dict*, *original_results*, *modified_results*)
Filters results for such ones that are unique across file changes

Parameters

- **original_file_dict** – Dict of lists of file contents before changes
- **modified_file_dict** – Dict of lists of file contents after changes
- **original_results** – List of results of the old files
- **modified_results** – List of results of the new files

Returns List of results from new files that are unique from all those that existed in the old changes

coalib.results.ResultFilter.**remove_range**(*file_contents*, *source_range*)
removes the chars covered by the sourceRange from the file

Parameters

- **file_contents** – list of lines in the file
- **source_range** – Source Range

Returns list of file contents without specified chars removed

coalib.results.ResultFilter.**remove_result_ranges_diffs**(*result_list*, *file_dict*)
Calculates the diffs to all files in file_dict that describe the removal of each respective result's affected code.

Parameters

- **result_list** – list of results
- **file_dict** – dict of file contents

Returns returnvalue[result][file] is a diff of the changes the removal of this result's affected code would cause for the file.

coalib.results.ResultFilter.**source_ranges_match**(*original_file_dict*, *diff_dict*, *original_result_diff_dict*, *modified_result_diff_dict*, *renamed_files*)
Checks whether the SourceRanges of two results match

Parameters

- **original_file_dict** – Dict of lists of file contents before changes
- **diff_dict** – Dict of diffs describing the changes per file
- **original_result_diff_dict** – diff for each file for this result
- **modified_result_diff_dict** – guess
- **renamed_files** – A dictionary containing file renamings across runs

Returns Boolean value whether the SourceRanges match

coalib.results.SourcePosition module

class coalib.results.SourcePosition.**SourcePosition**(*file: str*, *line=None*, *column=None*)
Bases: *coalib.results.TextPosition.TextPosition*

file

coalib.results.SourceRange module

class coalib.results.SourceRange.**SourceRange**(*start: coalib.results.SourcePosition.SourcePosition*, *end: (<class 'coalib.results.SourcePosition.SourcePosition'>, None) = None*)

Bases: *coalib.results.TextRange.TextRange*

expand(*file_contents*)

Passes a new SourceRange that covers the same area of a file as this one would. All values of None get replaced with absolute values.

values of None will be interpreted as follows: self.start.line is None: -> 1 self.start.column is None: -> 1 self.end.line is None: -> last line of file self.end.column is None: -> last column of self.end.line

Parameters **file_contents** – File contents of the applicable file

Returns TextRange with absolute values

file

classmethod from_absolute_position(*file: str*, *position_start: coalib.results.AbsolutePosition.AbsolutePosition*, *position_end: (<class 'coalib.results.AbsolutePosition.AbsolutePosition'>, None) = None*)

Creates a SourceRange from a start and end positions.

Parameters

- **file** – Name of the file.
- **position_start** – Start of range given by AbsolutePosition.
- **position_end** – End of range given by AbsolutePosition or None.

classmethod from_clang_range(*range*)

Creates a SourceRange from a clang SourceRange object.

Parameters **range** – A cindex.SourceRange object.

classmethod from_values(*file*, *start_line=None*, *start_column=None*, *end_line=None*, *end_column=None*)

renamed_file(*file_diff_dict: dict*)

Retrieves the filename this source range refers to while taking the possible file renamings in the given file_diff_dict into account:

Parameters **file_diff_dict** – A dictionary with filenames as key and their associated Diff objects as values.

coalib.results.TextPosition module

class coalib.results.TextPosition.**TextPosition**(*line: (<class 'int'>, None) = None*, *column: (<class 'int'>, None) = None*)

Bases: object

column

line

coalib.results.TextRange module

class coalib.results.TextRange.**TextRange**(*start: coalib.results.TextPosition.TextPosition*, *end: (<class 'coalib.results.TextPosition.TextPosition'>, None) = None*)

Bases: object

end

expand(*text_lines*)

Passes a new TextRange that covers the same area of a file as this one would. All values of None get replaced with absolute values.

values of None will be interpreted as follows: self.start.line is None: -> 1 self.start.column is None: -> 1 self.end.line is None: -> last line of file self.end.column is None: -> last column of self.end.line

Parameters **text_lines** – File contents of the applicable file

Returns TextRange with absolute values

classmethod from_values(*start_line=None*, *start_column=None*, *end_line=None*, *end_column=None*)

Creates a new TextRange.

Parameters

- **start_line** – The line number of the start position. The first line is 1.
- **start_column** – The column number of the start position. The first column is 1.
- **end_line** – The line number of the end position. If this parameter is None, then the end position is set the same like start position and end_column gets ignored.
- **end_column** – The column number of the end position.

Returns A TextRange.

classmethod join(*a*, *b*)

Creates a new TextRange that covers the area of two overlapping ones

Parameters

- **a** – TextRange (needs to overlap b)
- **b** – TextRange (needs to overlap a)

Returns A new TextRange covering the union of the Area of a and b

overlaps(*other*)

start

Module contents

1.1.9 coalib.settings package

Submodules

coalib.settings.ConfigurationGathering module

coalib.settings.ConfigurationGathering.**find_user_config**(*file_path*, *max_trials=10*)

Uses the filepath to find the most suitable user config file for the file by going down one directory at a time and finding config files there.

Parameters

- **file_path** – The path of the file whose user config needs to be found
- **max_trials** – The maximum number of directories to go down to.

Returns The config file's path, empty string if none was found

coalib.settings.ConfigurationGathering.**gather_configuration**(*acquire_settings*, *log_printer*, *arg_list=None*, *arg_parser=None*)

Loads all configuration files, retrieves bears and all needed settings, saves back if needed and warns about non-existent targets.

This function:

•Reads and merges all settings in sections from

–Default config

–User config

–Configuration file

–CLI

•Collects all the bears

•Fills up all needed settings

•Writes back the new sections to the configuration file if needed

•Gives all information back to caller

Parameters

- **acquire_settings** – The method to use for requesting settings. It will get a parameter which is a dictionary with the settings name as key and a list containing a description in [0] and the names of the bears who need this setting in all following indexes.
- **log_printer** – The log printer to use for logging. The log level will be adjusted to the one given by the section.
- **arg_list** – CLI args to use
- **arg_parser** – Instance of ArgParser that is used to parse none-setting arguments.

Returns

A tuple with the following contents:

- A dictionary with the sections
- Dictionary of list of local bears for each section
- Dictionary of list of global bears for each section
- The targets list

coalib.settings.ConfigurationGathering.**get_config_directory**(*section*)

Retrieves the configuration directory for the given section.

Given an empty section:

```
>>> section = Section("name")
```

The configuration directory is not defined and will therefore fallback to the current directory:

```
>>> get_config_directory(section) == os.path.abspath(".")
True
```

If the `files` setting is given with an originating coafile, the directory of the coafile will be assumed the configuration directory:

```
>>> section.append(Setting("files", "**", origin="/tmp/.coafile"))
>>> get_config_directory(section) == os.path.abspath('/tmp/')
True
```

However if its origin is already a directory this will be preserved:

```
>>> section['files'].origin = os.path.abspath('/tmp/dir/')
>>> os.makedirs(section['files'].origin, exist_ok=True)
>>> get_config_directory(section) == section['files'].origin
True
```

The user can manually set a project directory with the `project_dir` setting:

```
>>> section.append(Setting('project_dir', os.path.abspath('/tmp'), '/'))
>>> get_config_directory(section) == os.path.abspath('/tmp')
True
```

If no section is given, the current directory is returned:

```
>>> get_config_directory(None) == os.path.abspath(".")
True
```

To summarize, the config directory will be chosen by the following priorities if possible in that order:

- the `project_dir` setting
- the origin of the `files` setting, if it's a directory
- the directory of the origin of the `files` setting
- the current directory

Parameters **`section`** – The section to inspect.

Returns The directory where the project is lying.

`coalib.settings.ConfigurationGathering.`**`get_filtered_bears`**(*languages*, *log_printer*, *arg_parser=None*)

Fetch bears and filter them based on given list of languages.

Parameters

- **`languages`** – List of languages.
- **`log_printer`** – The log_printer to handle logging.
- **`arg_parser`** – An `ArgParser` object.

Returns Tuple containing dictionaries of local bears and global bears.

`coalib.settings.ConfigurationGathering.`**`load_config_file`**(*filename*, *log_printer*, *silent=False*)

Loads sections from a config file. Prints an appropriate warning if it doesn't exist and returns a section dict containing an empty default section in that case.

It assumes that the cli_sections are available.

Parameters

- **filename** – The file to load settings from.
- **log_printer** – The log printer to log the warning/error to (in case).
- **silent** – Whether or not to warn the user/exit if the file doesn't exist.

Raises **SystemExit** – Exits when the given filename is invalid and is not the default coafile. Only raised when `silent` is `False`.

coalib.settings.ConfigurationGathering.**load_configuration**(*arg_list*, *log_printer*, *arg_parser=None*)

Parses the CLI args and loads the config file accordingly, taking default_coafile and the users .coarc into account.

Parameters

- **arg_list** – The list of command line arguments.
- **log_printer** – The LogPrinter object for logging.

Returns A tuple holding (log_printer: LogPrinter, sections: dict(str, Section), targets: list(str)). (Types indicated after colon.)

coalib.settings.ConfigurationGathering.**merge_section_dicts**(*lower*, *higher*)

Merges the section dictionaries. The values of higher will take precedence over the ones of lower. Lower will hold the modified dict in the end.

Parameters

- **lower** – A section.
- **higher** – A section which values will take precedence over the ones from the other.

Returns The merged dict.

coalib.settings.ConfigurationGathering.**save_sections**(*sections*)

Saves the given sections if they are to be saved.

Parameters **sections** – A section dict.

coalib.settings.ConfigurationGathering.**warn_config_absent**(*sections*, *argument*, *log_printer*)

Checks if the given argument is present somewhere in the sections and emits a warning that code analysis can not be run without it.

Parameters

- **sections** – A dictionary of sections.
- **argument** – The argument to check for, e.g. "files".
- **log_printer** – A log printer to emit the warning to.

coalib.settings.ConfigurationGathering.**warn_nonexistent_targets**(*targets*, *sections*, *log_printer*)

Prints out a warning on the given log printer for all targets that are not existent within the given sections.

Parameters

- **targets** – The targets to check.
- **sections** – The sections to search. (Dict.)
- **log_printer** – The log printer to warn to.

coalib.settings.DocstringMetadata module

class coalib.settings.DocstringMetadata.**DocstringMetadata**(*desc*, *param_dict*, *retval_desc*)

Bases: object

classmethod from_docstring(*docstring*)

Parses a python docstring. Usable attributes are: :param @param :return @return

coalib.settings.FunctionMetadata module

class coalib.settings.FunctionMetadata.**FunctionMetadata**(*name: str*, *desc: str = ''*, *retval_desc: str = ''*, *non_optional_params: (<class 'dict'>, None) = None*, *optional_params: (<class 'dict'>, None) = None*, *omit: (<class 'set'>, <class 'tuple'>, <class 'list'>, <class 'frozenset'>) = frozenset()*, *deprecated_params: (<class 'set'>, <class 'tuple'>, <class 'list'>, <class 'frozenset'>) = frozenset()*)

Bases: object

add_deprecated_param(*original*, *alias*)

Adds an alias for the original setting. The alias setting will have the same metadata as the original one. If the original setting is not optional, the alias will default to None.

Parameters

- **original** – The name of the original setting.
- **alias** – The name of the alias for the original.

Raises **KeyError** – If the new setting doesn't exist in the metadata.

create_params_from_section(*section*)

Create a params dictionary for this function that holds all values the function needs plus optional ones that are available.

Parameters **section** – The section to retrieve the values from.

Returns The params dictionary.

desc

Returns description of the function.

filter_parameters(*dct*)

Filters the given dict for keys that are declared as parameters inside this metadata (either optional or non-optional).

You can use this function to safely pass parameters from a given dictionary:

```
>>> def multiply(a, b=2, c=0):
...     return a * b + c
>>> metadata = FunctionMetadata.from_function(multiply)
>>> args = metadata.filter_parameters({'a': 10, 'b': 20, 'd': 30})
```

You can safely pass the arguments to the function now:

```
>>> multiply(**args)  # 10 * 20
200
```

Parameters **dct** – The dict to filter.

Returns A new dict containing the filtered items.

classmethod **from_function**(*func*, *omit=frozenset()*)

Creates a FunctionMetadata object from a function. Please note that any variable argument lists are not supported. If you do not want the first (usual named 'self') argument to appear please pass the method of an actual INSTANCE of a class; passing the method of the class isn't enough. Alternatively you can add "self" to the omit set.

Parameters

- **func** – The function. If __metadata__ of the unbound function is present it will be copied and used, otherwise it will be generated.
- **omit** – A set of parameter names that are to be ignored.

Returns The FunctionMetadata object corresponding to the given function.

classmethod **merge**(**metadatas*)

Merges signatures of `FunctionMetadata` objects.

Parameter (either optional or non-optional) and non-parameter descriptions are merged from left to right, meaning the right hand metadata overrides the left hand one.

```
>>> def a(x, y):
...     '''
...     desc of *a*
...     :param x: x of a
...     :param y: y of a
...     :return:  5*x*y
...     '''
...     return 5 * x * y
>>> def b(x):
...     '''
...     desc of *b*
...     :param x: x of b
...     :return:  100*x
...     '''
...     return 100 * x
>>> metadata1 = FunctionMetadata.from_function(a)
>>> metadata2 = FunctionMetadata.from_function(b)
>>> merged = FunctionMetadata.merge(metadata1, metadata2)
>>> merged.name
"<Merged signature of 'a', 'b'>"
>>> merged.desc
'desc of *b*'
>>> merged.retval_desc
'100*x'
>>> merged.non_optional_params['x'][0]
'x of b'
>>> merged.non_optional_params['y'][0]
'y of a'
```

Parameters **metadatas** – The sequence of metadatas to merge.

Returns A `FunctionMetadata` object containing the merged signature of all given metadatas.

non_optional_params
Retrieves a dict containing the name of non optional parameters as the key and a tuple of a description and the python annotation. Values that are present in self.omit will be omitted.

optional_params
Retrieves a dict containing the name of optional parameters as the key and a tuple of a description, the python annotation and the default value. Values that are present in self.omit will be omitted.

str_nodesc = 'No description given.'

str_optional = "Optional, defaults to '{}'."

coalib.settings.Section module

class `coalib.settings.Section.`**Section**(*name*, *defaults=None*)
Bases: `object`

This class holds a set of settings.

add_or_create_setting(*setting*, *custom_key=None*, *allow_appending=True*)
Adds the value of the setting to an existing setting if there is already a setting with the key. Otherwise creates a new setting.

append(*setting*, *custom_key=None*)

bear_dirs()

copy()

Returns a deep copy of this object

delete_setting(*key*)
Delete a setting :param key: The key of the setting to be deleted

get(*key*, *default=''*, *ignore_defaults=False*)
Retrieves the item without raising an exception. If the item is not available an appropriate Setting will be generated from your provided default value.

Parameters

- **key** – The key of the setting to return.
- **default** – The default value
- **ignore_defaults** – Whether or not to ignore the default section.

Returns The setting.

is_enabled(*targets*)
Checks if this section is enabled or, if targets is not empty, if it is included in the targets list.

Parameters **targets** – List of target section names, all lower case.

Returns True or False

update(*other_section*, *ignore_defaults=False*)
Incorporates all keys and values from the other section into this one. Values from the other section override the ones from this one.

Default values from the other section override the default values from this only.

Parameters

- **other_section** – Another Section
- **ignore_defaults** – If set to true, do not take default values from other

Returns self

update_setting(*key*, *new_key=None*, *new_value=None*)
Updates a setting with new values. :param key: The old key string. :param new_key: The new key string. :param new_value: The new value for the setting

coalib.settings.Section.**append_to_sections**(*sections*, *key*, *value*, *origin*, *section_name=None*, *from_cli=False*)
Appends the given data as a Setting to a Section with the given name. If the Section does not exist before it will be created empty.

Parameters

- **sections** – The sections dictionary to add to.
- **key** – The key of the setting to add.
- **value** – The value of the setting to add.
- **origin** – The origin value of the setting to add.
- **section_name** – The name of the section to add to.
- **from_cli** – Whether or not this data comes from the CLI.

coalib.settings.SectionFilling module

coalib.settings.SectionFilling.**fill_section**(*section*, *acquire_settings*, *log_printer*, *bears*)
Retrieves needed settings from given bears and asks the user for missing values.

If a setting is requested by several bears, the help text from the latest bear will be taken.

Parameters

- **section** – A section containing available settings. Settings will be added if some are missing.
- **acquire_settings** – The method to use for requesting settings. It will get a parameter which is a dictionary with the settings name as key and a list containing a description in [0] and the names of the bears who need this setting in all following indexes.
- **log_printer** – The log printer for logging.
- **bears** – All bear classes or instances.

Returns The new section.

coalib.settings.SectionFilling.**fill_settings**(*sections*, *acquire_settings*, *log_printer*)
Retrieves all bears and requests missing settings via the given acquire_settings method.

This will retrieve all bears and their dependencies.

Parameters

- **sections** – The sections to fill up, modified in place.

- **acquire_settings** – The method to use for requesting settings. It will get a parameter which is a dictionary with the settings name as key and a list containing a description in [0] and the names of the bears who need this setting in all following indexes.
- **log_printer** – The log printer to use for logging.

Returns A tuple containing (local_bears, global_bears), each of them being a dictionary with the section name as key and as value the bears as a list.

coalib.settings.Setting module

class coalib.settings.Setting.**Setting**(*key*, *value*, *origin=''*, *strip_whitespaces=True*, *list_delimiters=(',', ';')*, *from_cli=False*, *remove_empty_iter_elements=True*)

Bases: coala_utils.string_processing.StringConverter.StringConverter

A Setting consists mainly of a key and a value. It mainly offers many conversions into common data types.

key

coalib.settings.Setting.**glob**(*obj*, **args*, ***kwargs*)

Creates a path in which all special glob characters in all the parent directories in the given setting are properly escaped.

Parameters **obj** – The Setting object from which the key is obtained.

Returns Returns a path in which special glob characters are escaped.

coalib.settings.Setting.**glob_list**(*obj*, **args*, ***kwargs*)

Creates a list of paths in which all special glob characters in all the parent directories of all paths in the given setting are properly escaped.

Parameters **obj** – The Setting object from which the key is obtained.

Returns Returns a list of paths in which special glob characters are escaped.

coalib.settings.Setting.**path**(*obj*, **args*, ***kwargs*)

coalib.settings.Setting.**path_list**(*obj*, **args*, ***kwargs*)

coalib.settings.Setting.**typed_dict**(*key_type*, *value_type*, *default*)

Creates a function that converts a setting into a dict with the given types.

Parameters

- **key_type** – The type conversion function for the keys.
- **value_type** – The type conversion function for the values.
- **default** – The default value to use if no one is given by the user.

Returns A conversion function.

coalib.settings.Setting.**typed_list**(*conversion_func*)

Creates a function that converts a setting into a list of elements each converted with the given conversion function.

Parameters **conversion_func** – The conversion function that converts a string into your desired list item object.

Returns A conversion function.

coalib.settings.Setting.**typed_ordered_dict**(*key_type*, *value_type*, *default*)

Creates a function that converts a setting into an ordered dict with the given types.

Parameters

- **key_type** – The type conversion function for the keys.
- **value_type** – The type conversion function for the values.
- **default** – The default value to use if no one is given by the user.

Returns A conversion function.

coalib.settings.Setting.**url**(*obj*, **args*, ***kwargs*)

Module contents

1.1.10 coalib.testing package

Submodules

coalib.testing.BearTestHelper module

coalib.testing.BearTestHelper.**generate_skip_decorator**(*bear*)
Creates a skip decorator for a *unittest* module test from a bear.

check_prerequisites is used to determine a test skip.

Parameters **bear** – The bear whose prerequisites determine the test skip.

Returns A decorator that skips the test if appropriate.

coalib.testing.LocalBearTestHelper module

class coalib.testing.LocalBearTestHelper.**LocalBearTestHelper**(*methodName='runTest'*)
Bases: unittest.case.TestCase

This is a helper class for simplification of testing of local bears.

Please note that all abstraction will prepare the lines so you don't need to do that if you use them.

If you miss some methods, get in contact with us, we'll be happy to help!

check_results(*local_bear*, *lines*, *results*, *filename=None*, *check_order=False*, *force_linebreaks=True*, *create_tempfile=True*, *tempfile_kwargs={}*, *settings={}*)
Asserts that a check of the given lines with the given local bear does yield exactly the given results.

Parameters

- **local_bear** – The local bear to check with.
- **lines** – The lines to check. (List of strings)
- **results** – The expected list of results.
- **filename** – The filename, if it matters.
- **force_linebreaks** – Whether to append newlines at each line if needed. (Bears expect a n for every line)
- **create_tempfile** – Whether to save lines in tempfile if needed.
- **tempfile_kwargs** – Kwargs passed to tempfile.mkstemp().
- **settings** – A dictionary of keys and values (both strings) from which settings will be created that will be made available for the tested bear.

check_validity(*local_bear*, *lines*, *filename=None*, *valid=True*, *force_linebreaks=True*, *create_tempfile=True*, *tempfile_kwargs={}*)

Asserts that a check of the given lines with the given local bear either yields or does not yield any results.

Parameters

- **local_bear** – The local bear to check with.
- **lines** – The lines to check. (List of strings)
- **filename** – The filename, if it matters.
- **valid** – Whether the lines are valid or not.
- **force_linebreaks** – Whether to append newlines at each line if needed. (Bears expect a n for every line)
- **create_tempfile** – Whether to save lines in tempfile if needed.
- **tempfile_kwargs** – Kwargs passed to tempfile.mkstemp().

coalib.testing.LocalBearTestHelper.**execute_bear**(*bear*, **args*, ***kwargs*)

coalib.testing.LocalBearTestHelper.**verify_local_bear**(*bear*, *valid_files*, *invalid_files*, *filename=None*, *settings={}*, *force_linebreaks=True*, *create_tempfile=True*, *timeout=None*, *tempfile_kwargs={}*)

Generates a test for a local bear by checking the given valid and invalid file contents. Simply use it on your module level like:

YourTestName = verify_local_bear(YourBear, (['valid line'],), (['invalid line'],))

Parameters

- **bear** – The Bear class to test.
- **valid_files** – An iterable of files as a string list that won't yield results.
- **invalid_files** – An iterable of files as a string list that must yield results.
- **filename** – The filename to use for valid and invalid files.
- **settings** – A dictionary of keys and values (both string) from which settings will be created that will be made available for the tested bear.
- **force_linebreaks** – Whether to append newlines at each line if needed. (Bears expect a n for every line)
- **create_tempfile** – Whether to save lines in tempfile if needed.
- **timeout** – The total time to run the test for.
- **tempfile_kwargs** – Kwargs passed to tempfile.mkstemp() if tempfile needs to be created.

Returns A unittest.TestCase object.

Module contents

1.2 Submodules

1.3 coalib.coala module

coalib.coala.**main**()

1.4 coalib.coala_ci module

coalib.coala_ci.**main**()

1.5 coalib.coala_delete_orig module

coalib.coala_delete_orig.**main**(*log_printer=None*, *section: coalib.settings.Section.Section = None*)

1.6 coalib.coala_format module

coalib.coala_format.**main**()

1.7 coalib.coala_json module

coalib.coala_json.**main**()

1.8 coalib.coala_main module

coalib.coala_main.**do_nothing**(**args*)

coalib.coala_main.**run_coala**(*console_printer=None*, *log_printer=None*, *print_results=<function <lambda>>*, *acquire_settings=<function fail_acquire_settings>*, *print_section_beginning=<function <lambda>>*, *nothing_done=<function <lambda>>*, *force_show_patch=False*, *arg_parser=None*, *arg_list=None*)

This is a main method that should be usable for almost all purposes and reduces executing coala to one function call.

Parameters

- **console_printer** – Object to print messages on the console.
- **log_printer** – A LogPrinter object to use for logging.
- **print_results** – A callback that takes a LogPrinter, a section, a list of results to be printed, the file dict and the mutable file diff dict.

- **acquire_settings** – The method to use for requesting settings. It will get a parameter which is a dictionary with the settings name as key and a list containing a description in [0] and the names of the bears who need this setting in all following indexes.
- **print_section_beginning** – A callback that will be called with a section name string whenever analysis of a new section is started.
- **nothing_done** – A callback that will be called with only a log printer that shall indicate that nothing was done.
- **force_show_patch** – If set to True, a patch will be always shown. (Using ApplyPatchAction.)
- **arg_list** – The CLI argument list.

Returns A dictionary containing a list of results for all analyzed sections as key.

1.9 coalib.coala_modes module

coalib.coala_modes.**mode_format**()

coalib.coala_modes.**mode_json**(*args*)

coalib.coala_modes.**mode_non_interactive**(*console_printer*, *args*)

coalib.coala_modes.**mode_normal**(*console_printer*, *log_printer*)

1.10 Module contents

The coalib package is a collection of various subpackages regarding writing, executing and editing bears. Various other packages such as formatting and settings are also included in coalib.

coalib.**assert_supported_version**()

coalib.**get_version**()

C

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Y