SublimeLinter Documentation

Release 4.0.0

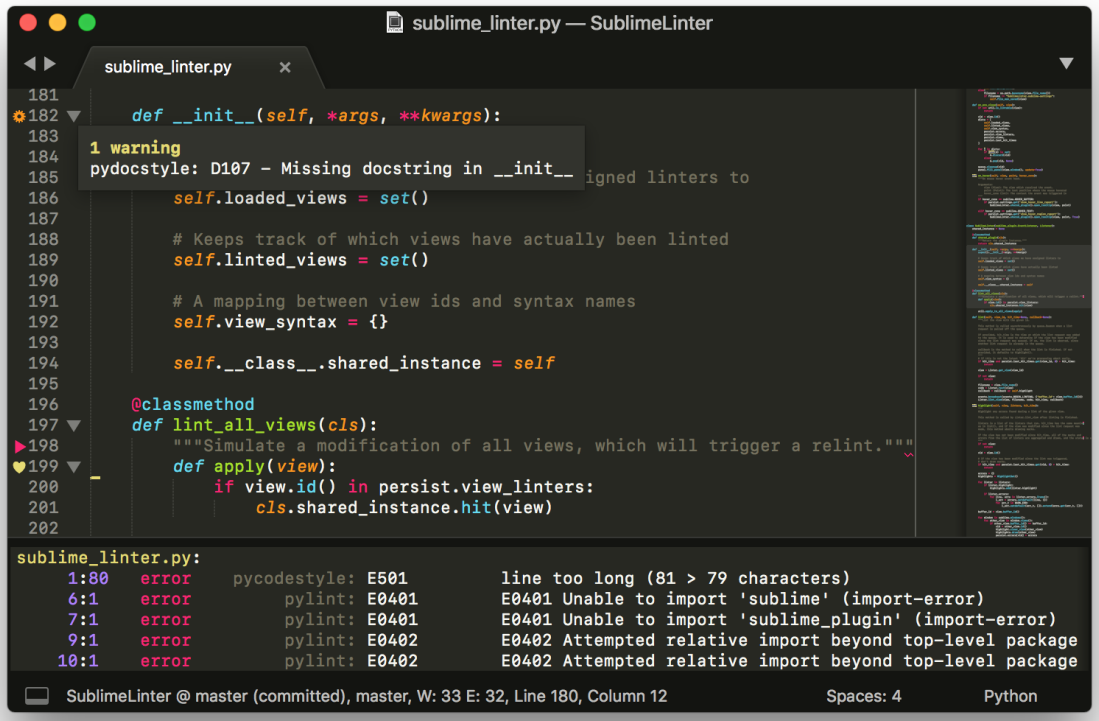
The SublimeLinter Community

May 20, 2018

User Documentation

SublimeLinter is a plugin for Sublime Text that provides a framework for linting code.

SublimeLinter repo and issues can be reached on GitHub.



CHAPTER 1

Installation

SublimeLinter and corresponding linter plugins should be installed using PackageControl.

First install PackageControl and then see its usage, which explains how to install packages to SublimeText.

Notice that it may be required to restart SublimeText after package installation.

Make sure to read SublimeLinter messages.

CHAPTER 2

Settings

The settings are documented in the default settings file, so you can refer to them while editing your settings.

Note: Settings are checked for correctness, a message will display with errors. You need to fix or remove incorrect settings, like typos and deprecated settings.

This page covers some extra tricks and how to work with project specific settings.

2.1 Settings stack

SublimeLinter merges settings from several sources to calculate the value. Settings are merged in the following order:

1. Default settings
2. User settings
3. *Project settings* (only "linters" settings)

2.2 Styles (colors)

Colors are applied to highlights and gutter icons using **scopes**.

Scopes are how Sublime Text manages color. Regions of code (and sections of the gutter) are labelled with scopes. You can think of scopes as class names in an HTML file. These scopes then receive color from the color scheme, which is kinda like a CSS stylesheet.

SublimeLinter expects the scopes `markup.warning` and `markup.error` to get correct colors from most color schemes. We use scopes like `region.redish` for color schemes that don't provide colors for these scopes.

To change the colors, you can use region.colorish scopes: redish, orangish, yellowish, greenish, bluish, purplish, pinkish

Or you can customize your color scheme.

2.3 Project settings

Only the "linters" settings in can be changed in a project. All other settings can only be changed in your user settings.

SublimeLinter project settings are defined by a `"SublimeLinter"` object within Sublime Text's sublime-project file.

Note: Read more about project setting in Sublime Text's documentation.

Here is an example project settings file where the flake8 linter has been disabled:

```
{
    "folders":
    [
        {
            "path": "."
        }
    ],
    "SublimeLinter":
    {
        "linters":
        {
            "flake8": {
                "disable": true
            }
        }
    }
}
```

Note: Do not put the `"SublimeLinter"` object inside a `"settings"` object, or anywhere else but directly in the root object of the sublime-project file.

2.4 Settings Expansion

After the settings have been merged, SublimeLinter iterates over all settings values and expands any strings. This uses Sublime Text's *expand_variables* API, which uses the `${varname}` syntax and supports placeholders (`${varname:placeholder}`). Placeholders are resolved recursively (e.g. `${XDG_CONFIG_HOME:$HOME/.config}`).

To insert a literal `$` character, use `\\$`.

The following case-sensitive variables are provided:

- `packages`
- `platform`
- `file`
- `file_path`

- `file_name`
- `file_base_name`
- `file_extension`
- `folder`
- `project`
- `project_path`
- `project_name`
- `project_base_name`
- `project_extension`
- all environment variables

Note: See the documentation on build systems for an explanation of what each variable contains.

We enhanced the expansion for `folder`. It now attempts to guess the correct folder if you have multiple folders open in a window.

Additionally, ~ will get expanded using os.path.expanduser.

CHAPTER 3

Linter Settings

Each linter plugin can provide its own settings. SublimeLinter already provides these for every linter:

3.1 args

Specifies extra arguments to pass to an external binary.

The value may be a string or an array. If it is a string, it will be parsed as if it were passed on a command line. For example, these values are equivalent:

```
{
    "args": "--foo=bar --bar=7 --no-baz"
}

{
    "args": [
        "--foo=bar",
        "--bar=7",
        "--no-baz"
    ]
}
```

The default value is an empty array.

3.2 disable

Disables the linter.

3.3 env

Set additional environment variables.

```
{
    "env": "{'GEM_HOME': '~/foo/bar'}"
}
```

3.4 excludes

This setting specifies a list of path patterns to exclude from linting. If there is only a single pattern, the value may be a string. Otherwise it must be an array of patterns.

Patterns are matched against a file's **absolute path** with all symlinks/shortcuts resolved. This means to match a filename, you must match everything in the path before the filename. For example, to exclude any python files whose name begins with "foo", you would use this pattern:

```
{
    "excludes": "*/foo*.py"
}
```

The default value is an empty array. Untitled views can be ignored with `<untitled>`, and you can use `!` to negate a pattern. Note that *Settings Expansion* can be used here as well.

3.5 executable

At any time you can manually set the executable a linter should use. This can be a string or a list.

```
{
    "executable": "${folder}/node_modules/bin/eslint",
    "executable": ["py", "-3", "-m", "flake8"],
    "executable": ["nvm", "exec", "8.9", "eslint"]
}
```

See *Settings Expansion* for more info on using variables.

3.6 lint_mode

Lint Mode determines when the linter is run. * *background*: asynchronously on every change * *load_save*: when a file is opened and every time it's saved * *manual*: only when calling the Lint This View command * *save*: only when a file is saved

3.7 python

This should point to a python binary on your system. Alternatively it can be set to a version, in which case we try to find a python binary on your system matching that version (using PATH).

It then executes `python -m script_name` (where script_name is e.g. `flake8`).

3.8 selector

This defines if when given linter is activated for specific file types. It should be a string containing a list of comma separated selectors.

For example, by default yamllint is activated only for YAML files (`source.yaml`) files. But we also want to activate it for ansible files, which have the `source.ansible` scope.

To do that, we can override the selector for this linter:

```
{
    "linters": {
        "yamllint":
        {
            "selector": "source.yaml, source.ansible"

        },
    }
}
```

To find out what selector to use for given file type, use the "Tools > Developer > Show Scope Name" menu entry.

It's also possible to exclude scopes using the - operator. E.g. to disable embedded code in situation where linting doesn't make sense. For eslint we disable linting in html script attributes:

```
{
    'selector': 'source.js - meta.attribute-with-value'
}
```

Note: The selector setting takes precedence over the deprecated *syntax* property.

3.9 styles

Styles can be set per linter.

You can change the color (via scope) or icon per linter, for errors or warnings, and even for each error code if the plugin reports them.

Example: this changes the appearance of shellcheck warnings:

```
{
    "shellcheck": {
        "styles": [
            {
                "mark_style": "stippled_underline",
                "scope": "region.bluish",
                "types": ["warning"]
            }
        ]
    }
}
```

Example: this changes the appearance of whitespace warnings in flake8:

```
{
    "flake8": {
        "styles": [
            {
                "mark_style": "outline",
                "scope": "comment",
                "icon": "none",
                "codes": ["W293", "W291", "W292"]
            }
        ]
    }
}
```

3.10 working_dir

This setting specifies the linter working directory. The value must be a string, corresponding to a valid directory path.

For example (this is also the default):

```
{
    "working_dir": "${folder:$file_path}"
}
```

Here the linter will get invoked from the `${folder}` directory or the file's directory if it is not contained within a project folder.

See *Settings Expansion* for more info on using variables.

CHAPTER 4

Troubleshooting

This page covers a number of common problems and how to debug them. If this doesn't help you, look in the GitHub issues for similar issues (also look in the closed issues).

If you end up opening a new issue, please include (relevant) settings and a *debug log*.

4.1 Debug mode

In debug mode, SublimeLinter prints additional information to Sublime Text's console. Among other things it will list if a linter was able to run and its output.

To enable this mode, set `"debug"` to `true` in your SublimeLinter settings.

4.2 The linter doesn't work!

When a linter does not work try to run the program from the command line (Terminal in Mac OS X/Linux, Command Prompt in Windows). If it does not work there, it definitely won't work in SublimeLinter.

Here are the most common reasons why a linter does not work:

- The syntax is a variation (e.g. `"html (django)"`) that isn't mapped to a known syntax (e.g. `"html"`). The detected syntax is printed to the console in debug mode. Also note that plugins should move to using the selector setting instead of the old syntaxes attribute. You can use the *selector* linter setting right now instead of the "syntax_map".
- The linter binary or its dependencies are not installed. Be sure to install the linter as documented in the linter plugin's README.
- The linter binary is installed, but its path is not available to SublimeLinter. Follow the steps in *Debugging PATH problems* below.

4.3 Debugging PATH problems

In order for SublimeLinter to use linter executables, it must be able to find them on your system. There are two possible sources for this information:

1. The PATH environment variable.
2. The `"paths"` setting.

In *debug mode* SublimeLinter prints the computed path to the console. If a linter's executable cannot be found, the debug output will include a `cannot locate <linter>` message.

A linter may have additional dependencies (e.g. NodeJS) that may be missing. The console should also have information about that.

We noticed some users having an issue where a linter couldn't find "node" even though "node" is in their $PATH. If you're having this problem you can remedy it by patching the "env" for that linter in your settings like so:

```
"linters": {
  "eslint": {
      "env": {"PATH":"/usr/local/bin/"}
  }
}
```

4.3.1 Finding a linter executable

If a linter executable cannot be found, these are steps you can take to locate the source of the problem.

First check if the executable is in your PATH. Enter the following at a command prompt, replacing `<linter>` with the correct name (e.g. `eslint`):

```
# Mac OS X, Linux
which <linter>

# Windows
where <linter>
```

If this fails to output the executable's location it will not work. Make sure the executable is installed and if necessary edit your PATH. How to edit your PATH strongly depends on you operating system and its specific configuration. The internet is full of HOWTO's on this subject.

4.3.2 Adding to the "paths" setting

If you cannot rely on the PATH environment variable, paths can be configured in SublimeLinter's settings.

For example, let's say you are using `rbenv` on macOS. To add the path `~/.rbenv/shims` you would change the `"paths"` setting like this:

```
"paths": {
    "linux": [],
    "osx": [
        "~/.rbenv/shims"
    ],
    "windows": []
}
```

CHAPTER 5

Creating a gutter theme

Use one of the existing gutter themes as a starting point. You can find them in the repo.

To colorize icons the .gutter-theme file should contain: `{ "colorize": true }`. In this case your icons should be mostly white, (with shades of gray).

If you set colorize to false, Sublime Text will still colorize them. To maintain the original color we colorize them using a scope that should get a white color: `region.whitish`. If this results in incorrectly colored icons, this scope needs to be added to your color scheme.

Gutter images are scaled to to 16 x 16. For best results with Retina displays, gutter images should be 32 x 32.

To install your theme place the directory in Packages/User.

CHAPTER 6

Creating a linter plugin

Fork the template repo to get started on your plugin. It contains a howto with all the information you need.

The SublimeLinter package control channel lists all existing plugins, you can find examples there too.

To publish your plugin, start a PR.

CHAPTER 7

Linter Attributes

All linter plugins must be subclasses of `SublimeLinter.lint.Linter`. The Linter class provides the attributes and methods necessary to make linters work within SublimeLinter.

The Linter class is designed to allow interfacing with most linter executables/libraries through the configuration of class attributes. Some linters, however, will need to do more work to set up the environment for the linter executable, or may do the linting directly in the linter plugin itself. In that case, refer to the *linter method documentation*.

7.1 cmd (mandatory)

A tuple or callable that returns a tuple, containing the command line (with arguments) used to lint.

- If `cmd` is `None`, it is assumed the plugin overrides the `run` method.
- A `${file}` argument will be replaced with the full filename, which allows you to guarantee that certain arguments will be passed after the filename.
- When *tempfile_suffix* is set, the filename will be the temp filename.
- A `${args}` argument will be replaced with the arguments built from the linter settings, which allows you to guarantee that certain arguments will be passed at the end of the argument list.

7.2 default_type

Usually the `error` and `warning` named capture groups in the *regex (mandatory)* classify the problems. If the linter output does not provide information which can be captured with those groups, this attribute is used to determine how to classify the linter error. The value should be `SublimeLinter.lint.ERROR` or `SublimeLinter.lint.WARNING`.

The default value is `SublimeLinter.lint.ERROR`.

7.3 defaults

Set this attribute to a dict of setting names and values to provide defaults for the linter's settings.

The most important setting is `"selector"`, which specifies the scopes for which the linter is run.

If a setting will be passed as an argument to the linter executable, you may specify the format of the argument here and the setting will automatically be passed as an argument to the executable. The format specification is as follows:

```
<prefix><name><joiner>[<sep>[+]]
```

- **prefix** – Either `@`, `-` or `--`.
- **name** – The name of the setting.
- **joiner** – Either `=` or `:`. If `prefix` is `@`, this attribute is ignored (but may not be omitted). Otherwise, if this is `=`, the setting value is joined with `name` by `=` and passed as a single argument. If `:`, `name` and the value are passed as separate arguments.
- **sep** – If the argument accepts a list of values, `sep` specifies the character used to delimit the list (usually `,`).
- **+** – If the setting can be a list of values, but each value must be passed as a separate argument, terminate the setting with `+`.

After the format is parsed, the prefix and suffix are removed and the setting key is replaced with `name`.

Note: When building the list of arguments to pass to the linter, if the setting value is `falsy` (`None`, zero, `False`, or an empty sequence), the argument is not passed to the linter.

7.4 error_stream

Some linters report problem on `stdout`, some on `stderr`. By default SublimeLinter listens for both. If that's wrong you can set this to `SublimeLinter.lint.STREAM_STDOUT` or `SublimeLinter.lint.STREAM_STDERR`.

Note however, it's important to capture errors generated by the linter itself, for example a bad command line argument or some internal error. Usually linters will report their own errors on `stderr`. To ensure you capture both regular linter output and internal linter errors, you need to determine on which stream the linter writes reports and errors.

7.5 line_col_base

This attribute is a tuple that defines the number base used by linters in reporting line and column numbers. In general, most linters use one-based line numbers and column numbers, so the default value is `(1, 1)`. If a linter uses zero-based line numbers or column numbers, the linter class should define this attribute accordingly.

Note: For example, if the linter reports one-based line numbers but zero-based column numbers, the value of this attribute should be `(1, 0)`.

7.6 multiline

This attribute determines whether the *regex (mandatory)* attribute parses multiple lines. The linter may output multiline error messages, but if `regex` only parses single lines, this attribute should be `False` (the default).

- If `multiline` is `False`, the linter output is split into lines (using `str.splitlines` and each line is matched against `regex` pattern.
- If `multiline` is `True`, the linter output is iterated over using `re.finditer` until no more matches are found.

Note: It is important that you set this attribute correctly; it does more than just add the `re.MULTILINE` flag when it compiles the `regex` pattern.

7.7 name

Usually the name of the linter is derived from the name of the class. If that doesn't work out, you can also set it explicitly with this attribute.

7.8 re_flags

If you wish to add custom `re flags` that are used when compiling the *regex (mandatory)* pattern, you may specify them here.

For example, if you want the pattern to be case-insensitive, you could do this:

```
re_flags = re.IGNORECASE
```

Note: These flags can also be included within the `regex` pattern itself. It's up to you which technique you prefer.

7.9 regex (mandatory)

A python regular expression pattern used to extract information from the linter's output. The pattern must contain at least the following named capture groups:

Name	Description
line	The line number on which the problem occurred
message	The description of the problem

If your pattern doesn't have these groups you must override the *split_match* method to provide those values yourself.

In addition to the above capture groups, the pattern should contain the following named capture groups when possible:

Name	Description
col	The column number where the error occurred, or a string whose length provides the column number
error	If this is not empty, the error will be marked as an error by SublimeLinter
warning	If this is not empty, the error will be marked as a warning by SublimeLinter
near	If the linter does not provide a column number but mentions a name, match the name with this capture group and SublimeLinter will attempt to highlight that name. Enclosing single or double quotes will be stripped, you may include them in the capture group. If the linter provides a column number, you may still use this capture group and SublimeLinter will highlight that text (stripped of quotes) exactly.
code	The corresponding error code given by the linter, if supported.

7.10 tempfile_suffix

This attribute configures the behaviour of linter executables that cannot receive input from `stdin`.

If the linter executable require input from a file, SublimeLinter can automatically create a temp file from the current code and pass that file to the linter executable. To enable automatic temp file creation, set this attribute to the suffix of the temp file name (with or without a leading `.`).

7.10.1 File-only linters

Some linters can only work from an actual disk file, because they rely on an entire directory structure that cannot be realistically be copied to a temp directory. In such cases, you can mark a linter as *file-only* by setting *tempfile_suffix* to `-`.

File-only linters will only run on files that have not been modified since their last save, ensuring that what the user sees and what the linter executable sees is in sync.

7.11 word_re

If a linter reports a column position, SublimeLinter highlights the nearest word at that point. By default, SublimeLinter uses the regex pattern `r'^([-\w]+)'` to determine what is a word. You can customize the regex used to highlight words by setting this attribute to a pattern string or a compiled regex.

CHAPTER 8

Linter Methods

The `Linter` class is designed to allow interfacing with most linter executables/libraries through the configuration of class attributes. Some linters, however, will need to set up the environment for the linter executable, or may do the linting directly in the linter plugin itself.

In those cases, you will need to override one or more methods. SublimeLinter provides a set of methods that are designed to be overridden.

8.1 cmd

```
cmd(self)
```

If you need to dynamically generate the command line that is executed in order to lint, implement this method in your `Linter` subclass. Return a tuple/list with separate arguments. The first argument in the result should be the full path to the linter executable.

8.2 split_match

```
split_match(self, match)
```

This method extracts the named capture groups from the *regex (mandatory)* and return a tuple of *match*, *line*, *col*, *error*, *warning*, *message*, *near*.

If subclasses need to modify the values returned by the regex, they should override this method, call `super().split_match(match)`, then modify the values and return them.

CHAPTER 9

PythonLinter class

If your linter plugin interfaces with a linter that is written in python, you should subclass from `SublimeLinter.lint.PythonLinter`.

By doing so, you get the following features:

- Use correct environment using a `python` setting.
- Automatically find an environment using `pipenv`

CHAPTER 10

RubyLinter class

If your linter plugin interfaces with a linter that is written in ruby, you should subclass from `SublimeLinter.lint.RubyLinter`.

By doing so, you get support for rbenv and rvm (via rvm-auto-ruby).

10.1 rbenv and rvm support

During class construction, SublimeLinter attempts to locate the gem and ruby specified in *cmd*.

The following forms are valid for the first argument of `cmd`:

```
gem@ruby
gem
ruby
```

If `rbenv` is installed and the gem is also under `rbenv` control, the gem will be executed directly. Otherwise `(ruby [, gem])` will be executed.

If `rvm-auto-ruby` is installed, `(rvm-auto-ruby [, gem])` will be executed.

Otherwise `ruby` or `gem` will be executed.