External formatters

Stefan van der Haven | 12,406 installs | ★★★★★ (1) | Free

Use any program as formatter for your source files

Installation

Launch VS Code Quick Open (Ctrl+P), paste the following command, and press enter.

```
ext install SteefH.external-formatters
```

Copy More Info

Overview Version History Q & A Rating & Review

External Formatters

This extension lets you use an external command as formatter for your code. The command should take a document's content at `stdin` and return the formatted code at `stdout`.

Formatting trigger

When a formatter for a language or a glob pattern is configured, it will be used when the `Format Document` command is run from the command palette, or when the document is saved while the `editor.formatOnSave` setting is set to `true`.

Working directory

- When formatting an existing file, the command is run from the directory where the file resides.
- When the file is not saved yet, the command is run from the root directory of your workspace.
- When neither directory is available, the command will use the directory where Visual Studio Code is running.

Configuration

Formatting based on language ID

In your `settings.json`, add the following setting:

Categories

Formatters

Works with

Universal

Resources

Repository
License
Changelog
Download Extension

Project Details

SteefH/vscode-external-formatters
Last Commit: 4 years ago
3 Pull Requests
2 Open Issues

More Info

Version	0.2.0
Released on	23.12.2019, 17:23:22
Last updated	10.01.2020, 22:07:32
Publisher	Stefan van der Haven
Unique Identifier	SteefH.external-formatters
Report	Report Abuse

```
"externalFormatters.languages": {
    "<language id>": {
        "command": "<command to run>",
        "arguments": [
            "<first argument>",
            "<second argument>",
            ...
        ]
    },
    ...
}
```

`arguments` key is optional.

So, for instance, for `Terraform` code, you can use these settings:

```
"externalFormatters.languages": {
    "terraform": {
        "command": "terraform",
        "arguments": [
            "fmt",
            "-"
        ]
    }
}
```

You can define formatters for multiple languages, just add multiple entries under the `externalFormatters.languages` configuration key:

```
"externalFormatters.languages": {
    "foolang": {
        "command": "foofmt",
        "arguments": [
            "arg1",
            "arg2",
            "arg3"
        ]
    },
    "barlang": {
        "command": "format-bar"
    }
}
```

Formatting based on glob patterns

In your `settings.json`, add the following setting:

```
"externalFormatters.globPatterns": {
    "<glob pattern>": {
        "command": "<command to run>",
        "arguments": [
            "<first argument>",
            "<second argument>",
            ...
        ]
    },
```

```
    ...
}
```

<glob pattern> should be a valid glob pattern, with the following syntax:

- * to match one or more characters in a path segment
- ? to match on one character in a path segment
- ** to match any number of path segments, including none
- {} to group conditions (e.g. **/*.{ts,js} matches all TypeScript and JavaScript files)
- [] to declare a range of characters to match in a path segment (e.g., example.[0-9] to match on example.0, example.1, …)
- [!...] to negate a range of characters to match in a path segment (e.g., example.[!0-9] to match on example.a, example.b, but not example.0)

Note: a backslash (\) is not valid within a glob pattern, make sure to convert any backslash to slash when creating the glob pattern.

Here, The arguments key is optional as well, and multiple glob patterns are also supported.

TODO

- Formatter-specific working directory override
- Variable substitution in formatter settings (for instance: "${workspaceFolder}/tools/format-my-blub-file.sh")