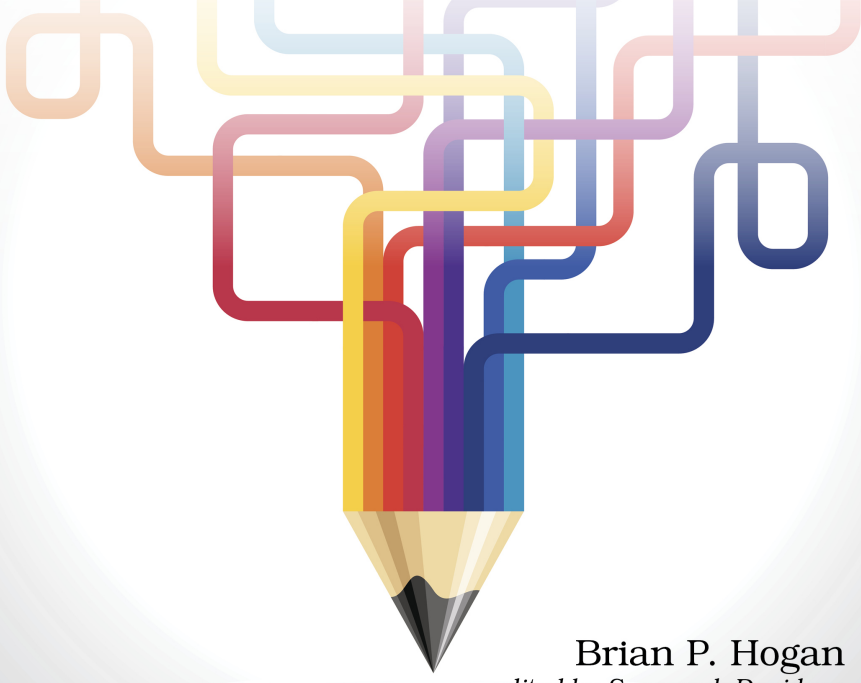
The Pragmatic Programmers

Pragmatic exPress

Write Better with Vale

Automate Your Style Guides and Lint Prose Like You Lint Code



Brian P. Hogan

edited by Susannah Davidson

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Ensuring Something Exists in the Document

The existence check flags things it finds, but you might want to ensure some phrase, word, or heading exists one or more times in text. Depending on what you're looking to do, you have two options for this: occurrence or conditional.

The occurrence check lets you specify the number of times you want a word, phrase, or character to occur in a given scope. It does this by counting the number of times the token value occurs within the scope. The conditional check lets you check that if one token exists, another does as well.

Ensuring Something Occurs Within a Range

In the previous chapter you added the Readability package to your style which scored your content based on several readability formulas. Sentence and paragraph length can also have an effect on readability, and you can use the occurrence check to count words in sentences or paragraphs.

Add the file styles/AwesomeCo/ParagraphLength.yml and add the following rule that checks paragraphs to ensure they are between 100 and 200 characters.

build_your_style/rules/styles/AwesomeCo/ParagraphLength.yml

```
extends: occurrence
message: "Try to keep paragraphs between 100 and 200 words. (%s)."
scope: paragraph
level: suggestion
min: 100
max: 200
token: \b(\w+)\b
```

This rule looks at each paragraph on its own and applies the regular expression in the token field. Since the expression looks for words, each word is a match. As long as the number of matches in that scope is between the min and max values, there won't be any errors raised.

None of the content you have in your site has large paragraphs, but 100 to 200 words is a good metric for general purpose writing. Run this rule against your files to ensure it works, and then turn it off in your .vale.ini configuration:

build_your_style/rules/.vale.ini

```
StylesPath = styles
MinAlertLevel = suggestion
Packages = Google, Microsoft, Readability, alex

[*.md]
BasedOnStyles = AwesomeCo
AwesomeCo.ParagraphLength = NO
```

You can always turn the rule on later, or use it in a different configuration.

When working with the occurrence check and a specific scope like heading or paragraph, remember that Vale looks for matches within each scope, not at the whole document. When you specify scope: paragraph, the rule applies to each paragraph, not across all paragraphs. Think of it as looping over each paragraph and executing the check. It's unaware of anything outside of that scope during the check. This means that you can use it to test for punctuation in a heading, for example, but you can't use it to ensure a specific heading occurs once in your document. You'll have to use a different scope for that, which you'll do later in Writing Custom Rules with Scripts, on page ?.

Ensuring Something Exists Based on Something Else

The conditional check lets you look for one pattern based on another. The most common use for this is the one Vale outlines in its own documentation: checking to see if an acronym or abbreviation has a definition first.

Add this rule to your project. Create styles/AwesomeCo/FirstUse.yml with the following rule that looks for an acronym of three to five characters, and then looks to see if that same acronym exists wrapped in parentheses:

build_your_style/rules/styles/AwesomeCo/FirstUse.yml

```
extends: conditional
message: "Define acronyms and abbreviations on first use. ('%s')"
ignorecase: false
level: suggestion
first: '\b([A-Z]{3,5}s?)\b'
second: '\(([A-Z]{3,5}s?)\)'
exceptions:
  - HTML
  - JSON
  - URL
  - ZIP
  - YAML
```

This rule assumes that you'll define terms with the spelled-out version, followed by the acronym or abbreviation in parentheses.

This rule uses the exceptions key to specify some tokens that won't trigger the rule. In this case, you don't need to define HTML and JSON first. If you have other known acronyms or abbreviations, you can add them as exceptions here as well.

Ensuring Something Exists in the Entire Document

Vale is markup-aware, meaning that it does its best to extract the text from Markdown, HTML, and other formats, while ignoring things like YAML front matter, inline code, and code blocks, as well as HTML tags and scripts. There will be times when you will want to use Vale to look at certain things that it skips.

For example, you may want to make sure that each of your tutorials has a `Conclusion` heading. You'd create a rule based on the `occurrence` check for that, but since Vale strips the markup out, you won't be able to identify headings. If you changed the `scope` to look only at headings, Vale would check each heading one at a time rather than looking at all headings in the document.

To solve this, use `scope: raw`. This instructs Vale to look at the entire raw document without processing it first. This means your rule will see every character, including code, markup, and special characters.

Test it out. Create a new file called `styles/AwesomeCo/Conclusion.yml` with this content:

build_your_style/rules/styles/AwesomeCo/Conclusion.yml

```
extends: occurrence
message: "A level 2 Conclusion heading must exist."
ignorecase: true
scope: raw
level: error
min: 1
max: 1
token: "#{2,} Conclusion"
```

Run Vale against `awk.md` and the first error you see is about conclusions:

```
$ vale awk.md

 awk.md
 1:1    error       A level 2 Conclusion heading    AwesomeCo.Conclusion
                    must exist.
...
```

This particular rule looked at the whole document for a specific pattern so Vale wasn't able to tell you what line number and column where the violation occurred, so it'll be at the top of the list.

You can get the position information when you use raw scopes. For example, you can create a rule that looks at the text of links for the words "click here" or "here" and flag them. Since Vale only looks at prose, it won't be able to

find hyperlinks, so you can use scope: raw with existence checks and a regular expression that looks for Markdown links.

Add the file styles/AwesomeCo/LinkHere.yml to your project:

build_your_style/rules/styles/AwesomeCo/LinkHere.yml

```
extends: existence
message: "Don't use 'here' or 'click here' as the content of a link."
ignorecase: true
scope: raw
level: error
raw:
  - '\[(click )?here\]\(.+\)'
```

Run Vale against sed.md and you'll find that the last line of the file has a link with that wording:

```
$ vale sed.md

 sed.md
 ...
 85:181  error  Don't use 'here' or 'click      AwesomeCo.LinkHere
                here' as the content of a
                link.
```

When you use the built in checks with the raw scope, you can't selectively turn those rules off within your document. That's because Vale isn't parsing the document and looking for those comments.

You can exert even more control over how Vale processes things by creating rules that include logic. You'll explore that in Writing Custom Rules with Scripts, on page ?.

Next you'll look at ensuring consistent capitalization.