International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 05 Issue: 04 | Apr-2018 www.irjet.net p-ISSN: 2395-0072

Customization of text editor: R language support plugin for atom text editor

Dona Mariya Shaju[1], Ruby Reetha George[2], Shreya Jacob[3], Linda Sara Mathew[4]

[1,2,3] Dept. of Computer Science and Engineering, MACE, Kerala, India
[4] Assistant Professor, Dept. of Computer Science and Engineering, MACE, Kerala, India

---

Abstract - *The paper deals with providing support for R language in Atom. Atom is a 21st century text editor that is hackable to the core. Atom comes by default with most of the features we need to develop a project. Customization in done in the form of plugins. Currently, atom doesn't provide much support for R language. R language is an open source programming language. R is a very flexible tool for doing mathematical and statistical analysis. A support for this language will be highly appreciated since R programming is used in machine learning, which is getting popular these days. A plugin to support R language is proposed to be built, with the features like syntax highlighting, code completion, error checking and supporting theme.*

***Key Words*: Auto-completion, Syntax highlighting, Linter, plugin, snippet.**

1. INTRODUCTION

Software development involves many different activities, and these activities have led to a variety of tools to support them. One of the early tools is still widely used today is the text editor. Text editors have a very general interface and provide primitive but general support for creating programs in traditional textual programming languages. The general research interest is in code reuse, and text editors provide support for reuse through mechanisms for textual cut-and paste. This support is very primitive, but it is heavily relied on by programmers to reuse old code and adapt it to new circumstances. This project deals with auto-completion, syntax highlighting and error checking for R language for atom text editor.

These efforts can lead to improved text editors. Text editing is an important user interface element, and editors are important because users spend significant time and effort working within them, even in single sessions. Understanding and improving text editors is therefore a worthwhile objective. Beyond editors, the aim is to better understand how programming languages and programs should support re-usability.

R is the most comprehensive statistical analysis package available. It incorporates all of the standard statistical tests, models, and analyses, as well as providing a comprehensive language for managing and manipulating data. New technology and ideas often appear first in R. R is a programming language and environment developed for statistical analysis by practicing statisticians and researchers.

Atom is a free and open-source text and source code editor for macOS, Linux, and Microsoft Windows with support for plug-ins written in Node.js, and embedded Git Control, developed by GitHub. It can also be used as an integrated development environment (IDE). Plugins are those features which make the text editors so special. These plugins add additional features to the existing editor. In short, we have full control over what the editor looks like and how it works. We can truly make it our own. That's the reason why its makers stress that Atom is "hackable".

1.1 Features Implemented

Autocompletion: Autocomplete, or word completion, is a feature in which an application predicts the rest of a word a user is typing. Autocomplete speeds up human-computer interactions when it correctly predicts the word a user intends to enter after only a few characters have been typed into a text input field. It works best in domains with a limited number of possible words (such as in command line interpreters), when some words are much more common (such as when addressing an e-mail) or writing structured and predictable text (as in source code editors).

Syntax-Highlighting: Syntax highlighting is a feature of text editors that are used for programming, scripting, or markup languages, such as HTML. The feature displays text, especially source code, in different colors and fonts according to the category of terms. This feature facilitates writing in a structured language such as a programming language or a markup language as both structures and syntax errors are visually distinct. Highlighting does not affect the meaning of the text itself; it is intended only for human readers.

Error-correction: Error correction mechanism helps in detecting the syntax rules and helps the programmer correct it. This increases the throughput. Brace matching, also known as bracket matching or parentheses matching, is also a feature of certain text editors and integrated development environments.

2. METHODOLOGY

2.1 Autocompletion

Snippets are an incredibly powerful way to quickly generate commonly needed code syntax from a shortcut.

The basic snippet format is as follows:

'source.js': 'console.log'

'prefix': 'log'

'body': 'console.log (${1:" crash"}); $2'

1. Load the snippets.cson from ~/.atom by selecting the Atom > Snippets menu.

2. Look for the "Scope" string.

3. Use $ followed by a number to mark a tab stop.

The leftmost keys are the selectors where these snippets should be active. The top-level snippet key is prepended by a period.

The next level of keys are the snippet names. These are used for describing the snippet in a more readable way in the snippet menu.

Under each snippet name is a prefix that should trigger the snippet and a body to insert when the snippet is triggered.

The above example adds a log snippet that would expand to: console.log("crash");

The string "crash" would be initially selected and pressing tab again would place the cursor after the;

2.2 Syntax Highlighting

Atom has to identify the language. This can be achieved by the three entries in the newly created CSON file:

'fileTypes': ['R', 'r']

'name': 'R'

'scopeName': 'source.r'

1. Create a subfolder named grammars to add new syntax rules.

2. Create a new CSON file named after the language to support, which is going to include all syntax highlighting rules.

3. Type in similar rules as below:

Syntax Rules

All the rules for the language was declared in a fourth entry: patterns. Any added rule was declared between {} in the patterns array:

'patterns': [

{ # First rule },

{ # Second rule },

...

{ # Last rule }

]

Matching a Syntax Element

{

'match': '\\b((0(x|X)[0-9a-fA-F]*)|(([0-9]+\\.?[0-9]*)|(\\.[0-9]+))((e|E)(\\+|-)?[0-9]+)?)(i|L|l|UL|ul|u|U|F|f|ll|LL|ull|ULL)?\\b'

'name': 'constant.numeric.r'

}

This rule detects any valid number. The match entry is filled with a valid regex. Then, each time Atom sees text matching this regex, it will encapsulate it in a span element with the class names indicated in name.

4. Several class names are added, each one separated by a dot. Any class name can be added in the name entry. There are some conventions to follow. Generally, the type of element you want to highlight is indicated and finished with the name of the language.

5. Match the beginning and end of a rule.

Sometimes it's easier to detect the beginning and the end of an element than the element itself. It's true for strings, for instance:

{

'begin': '\''

'beginCaptures':

'0':

'name': 'punctuation.definition.string.begin.r'

'end': '\''

'endCaptures':

'0':

'name': 'punctuation.definition.string.end.r'

'name': 'string.quoted.single.r'

'patterns': [

{

'match': '\\\\.'

'name': 'constant.character.escape.r'

```
    }
  ]
}
```

With this rule, Atom will search for a first quote to begin the string. Then, the next quote it finds will be the end of the string, as expected (the search is ungreedy).

6. Indicate all the regexes you want as begin and end delimiters. As always, the name entry must be filled with the class names you want for the whole retrieved element.

2.3 Error correction

A linter or lint refers to tools that analyze source code to flag programming errors, bugs, stylistic errors, and suspicious constructs. The term is originated from a Unix utility that examined C language source code.

Linter works while you're typing and checks if there are any syntax errors. If one forgets to put semicolon at the end of a line, linter will immediately show the error. It will save a lot of time. Not only that, it will force to write code more carefully.

The steps for creating linter in atom are as follows.

1. Bring up the Command Palette and type plugin. Among the commands select create new plugin.
2. Then an appropriate name is chosen.
3. Select the language for which linter is being created
4. The plugin directory will be opened in atom. It can be now modified according to the needs.

2.4 Theme customization

1. Firstly, Cmd+Shift+P is pressed and "Generate Syntax Theme" is typed to generate new theme package. Select "Generate Syntax Theme," and the path where the theme will be created is specified.
2. styles/colors.less is opened to change the various color variables which have already been defined. For example, turn @red into #f4c2c1.
3. styles/base.less is opened to modify the various selectors that have already been defined. These selectors style different parts of code in the editor such as comments, strings and the line numbers in the gutter.
4. Reload Atom by pressing Alt+Cmd+Ctrl+L to see the changes made reflected in the Atom window.

3. CONCLUSION

The text editor was customized to our needs. The theme was customized and other features were added. This customization can be extended to other programming languages as well.

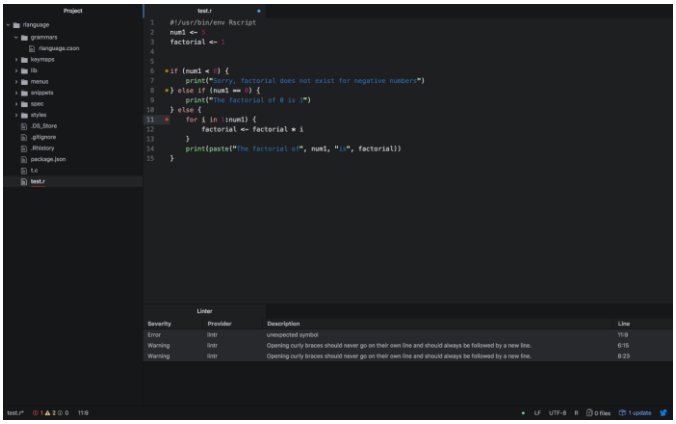

Fig -1: The customized Text Editor

REFERENCES

[1] A. A. Khwaja and J. E. Urban. Syntax-directed editing environments: Issues and features. In E. Deaton, K. M. George, H. Berghel, and G. Hedrick, editors, Proceedings of the ACMISIGAPP Symposium on Applied Computing, pages 230-237, Indianapolis, IN, Feb. 1993. ACM Press.

[2] V. Donzeau-Gouge et al., "A Structure-Oriented Program Editor: a First Step Towards Computer-Assisted Programming," Proc. of Int. Computing Symp., An- tibes, June 1975.

[3] M. E. Lesk, "Lex-A Lexical Analyzer Generator," Computer Science Technical Report '39, Bell Laboratories, Murray Hill, 1975.

[4] From homepage of http://flight-manual.atom.io/using-atom/sections/snippets/

[5] From homepage of http://www.sitepoint.com/how-to-write-a-syntax-highlighting-package-for-atom/