RHOLANG VSCODE INTEGRATION

DYLON T. EDWARDS

Abstract. Microsoft's Language Server Protocol (LSP) has standardized the mechanisms by which language-related tools communicate with editors. By supporting LSP, editors may be easily extended for arbitrary languages and features.

This document describes how Rholang (and soon, MeTTa IL) integrates with VSCode by way of an LSP server. Its concepts and features are discussed and a tutorial is presented that demonstrates how it may be installed and used.

1. Introduction

The Language Server Protocol (LSP) was originally developed by Microsoft for VSCode. Realizing its potential, Red Hat and Codenvy collaborated with Microsoft to make it an open standard.[5]

Since the most common editor features have been added to the API spec, nearly all modern editors include support for LSP and most language tools have been migrated to it. This allows them to communicate over a common API which reduces duplication of effort and avoids the need for individual translation layers. We take advantage of these to simplify the how we integrate with VSCode and other editors.

There are frameworks for building LSP-based language servers in nearly every major language. Using an existing framework eliminates most of the boilerplate logic we must support. We do not need to design our own protocol or add much logic to the client to integrate with our server. Specifying how to initialize the language server is nearly all the extension needs – VSCode has first-class support for LSP so it knows how to communicate with the server without further instruction. Most modern editors have similar support for LSP and can use the same language server without the need for specialization.

We selected Rust as the target language for the language server for a number of reasons, including:

- **Systems programming language ::** Rust is intended to write low-level software that interacts closely with hardware and the operating system, while maintaining fine-grained control over performance and resource management.
- **Memory safety ::** In Safe Rust, smart pointers statically ensure correctness in terms of memory access. There is no notion of a null pointer and all data structure accesses are bound checked so accessing unallocated memory is made difficult.
- **Reference counting ::** Rust does not directly support garbage collection but instead uses reference counting to deallocate objects. This has the benefit that objects are deallocated as soon as they fall out of scope, but also has the caveat that one must be mindful of cyclic graphs since their cycles must be unlinked before they can be deallocated.
- **Thread safety ::** The Rust compiler enforces design patterns that are understood to be thread-safe for the majority of use cases. This makes it difficult to introduce bugs related to race conditions or deadlocks.
- **Type safety ::** Rust is strongly typed which is to say that all terms in an expression must be of the same type. This seems tedious but reduces the likelihood of bugs related to unexpected type coercion.
- **High performance ::** Rust code is compiled ahead-of-time (AOT) to a native executable and has a lightweight, modular runtime so it has no issues with cold start times.
- **Direct hardware access ::** Rust facilitates interacting directly with hardware which is necessary for implementing device drivers.

Date: May 20, 2025.

- **Testing and dependency management ::** Rust has first-class support for automated testing and dependency management. This avoids the need for third-class frameworks and ensures they will always have strong support.

2. Architecture

The Rholang (LSP) Language Server acts as the glue between a standalone instance of RNode and the VSCode extension (the LSP- or language client). Communication between RNode and the extension is mediated by the language server.

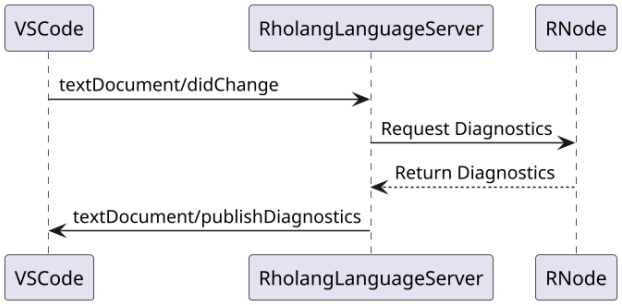

Figure 1. Sequence diagram showing how diagnostics are reported to VSCode from RNode when a document is edited.

3. Installation

The VSCode extension has two dependencies: (1) RNode, compiled with the LSP API[2], and (2) Rholang Language Server[3]. Once they have been installed, you may install the VSCode extension via either the VSCode Marketplace[1] or by cloning, building, and installing it from its GitHub repository[4].

The following is one method of installing RNode but there are others (like installing it from Docker Hub or via your Linux distribution's package manager). Please refer to its repository for more details.[2] To build and install RNode from source, please do the following:

```
git clone https://github.com/f1R3FLY-io/f1r3fly.git
cd f1r3fly
export SBT_OPTS="-Xmx4g -Xss2m -Dsbt.supershell=false"
sbt clean bnfc:generate compile stage
# Optional: Add `rnode` to your $PATH:
export PATH="$PWD/node/target/universal/stage/bin:$PATH"
```

Similarly, to install the Rholang Language Server, please do the following:

```
git clone https://github.com/F1R3FLY-io/rholang-language-server.git
cd rholang-language-server
cargo build
# Optional: Add `rholang-language-server` to your $PATH:
export PATH="$PWD/target/debug:$PATH"
```

If you wish to build and install the VSCode extension from source (such as for development), please do the following. Otherwise, please install it from the VSCode Marketplace[1].

```
git clone https://github.com/F1R3FLY-io/rholang-vscode-client.git
cd rholang-vscode-client
```

```
npm install
# Optional: Delete previous build artifacts if you've built the extension before
rm -rf out f1r3fly-io-rholang-*.vsix
npx vsce package
# Optional: Uninstall the extension if you have it installed
code --uninstall-extension f1r3fly-io.f1r3fly-io-rholang
code --install-extension f1r3fly-io-rholang-*.vsix
```

4. Initialization

Once you have installed RNode, you must initialize it in standalone mode for the language server and extension to work:

```
# NOTE: It is assumed that the path to the bin/ directory containing `rnode` is
# on your $PATH
rnode run --standalone
```

You may check whether the instance is running with the following command:

```
curl -s http://localhost:40403/status | jq
```

If it is running, the output should look something like the following:

```
{
  "address":
  ↪  "rnode://1f8709791aee6f4ba9ee45ab5f16e118a12b6ab9@99.104.196.8?protocol=40400&discovery=40404",
  "version": "RChain Node 1.0.0-SNAPSHOT (c5295563182e057950a0ee9f2e7bac66a1eedf03)",
  "peers": 0,
  "nodes": 0
}
```


5. Usage

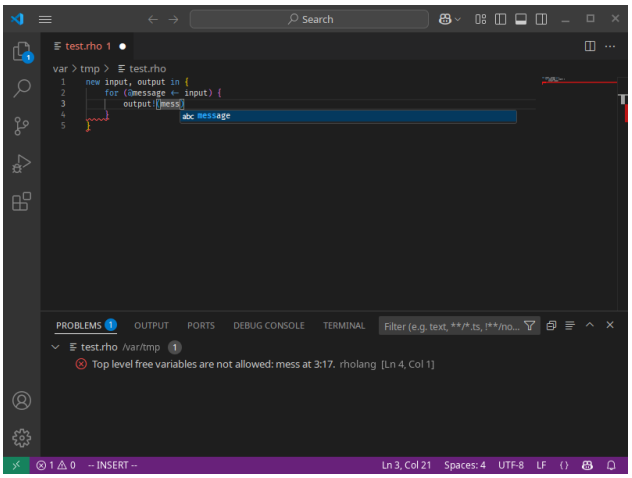

Figure 2. Editing a Rholang document with VSCode.

With the extension installed and RNode running in standalone mode, open a Rholang document and begin editing. A couple things you should notice are that it has syntax highlighting and rudimentary code-completion. It also supports automatic indentation and highlights the brackets surrounding the cursor's current context.

As you edit the document, you should notice that it provides error and warning diagnostics from the Rholang compiler (the compiler is executed by RNode). If you do not notice any diagnostics, then please ensure `rholang-language-server` is either on your `$PATH` or that you have configured the path to it in your settings (as described next).

To edit the extension's settings, do the following:

(1) Open the `Extensions` panel (`Ctrl+Shift+X`).
(2) Next to `Rholang (F1R3FLY.io)`, click the gear icon (`Manage`) followed by the `Settings` context menu item.

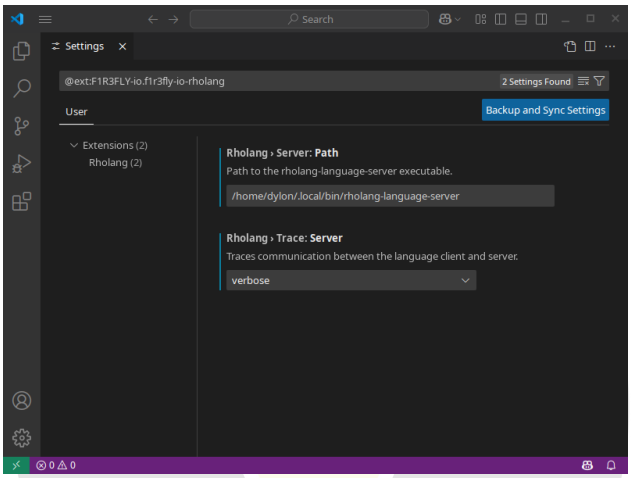

FIGURE 3. Rholang extension settings.

Within the settings, you may configure the path to your `rholang-language-server` and edit other extension-related settings. Presently, there are not many settings but it will become much more configurable in the near future.

6. FEATURES

LSP support for Rholang is still in its infancy, but the following are supported:

- **Syntax Highlighting ::** Regex-based text highlights.
- **Automatic Indentation ::** Regex-based, block-scoped indentation rules.
- **Diagnostics ::** When documents are opened or edited, they are validated and VSCode is notified of any error or warning messages.
- **Code Completion ::** Symbols contained within opened documents are made available for completion.
- **Language Server Path Configuration ::** Allows the path to the language server to be specified if it is not on the `$PATH`.

6.1. Upcoming Features.

Soon, the following will be added:

- **Goto Definition ::** Jump to the definition of a process symbol.
- **Find References ::** Locate all the Rholang file locations where a process symbol is used.
- **Call Hierarchies ::** A graph of incoming and outgoing channels within the context of a process. This can be used to construct and analyze dependency graphs.

- **Document Highlighting ::** Highlighting of relevant regions of the current document as it is being edited.
- **Hover Previews ::** Information about process symbols as they are hovered over.
- **Recommended Actions ::** Recommendations about resolving errors and warnings.
- **Code Folding ::** Ability to hide various regions of the current document.
- **Hierarchical Selections ::** Contextual selection of text around the cursor based on hierarchical scope.
- **Document Symbols ::** A navigable list of process symbols within the current document.
- **Semantic Highlighting ::** Semantic-based highlighting (based on the AST and not the results of a regex).
- **Contextual Code Completions ::** Code completions based on the available process symbols within the current scope.
- **Process Signatures ::** Synopses about processes based on their free parameters.
- **Formatting ::** Whole- and partial-document (selection-based) formatting and auto-formatting on save.
- **Symbol Renaming ::** Ability to rename a symbol across the workspace.
- **Workspace Symbols ::** Navigable list of process symbols within the workspace (global scope).
- **Code Evaluation ::** Ability to evaluate whole documents or partial selections within the context of a standalone RSpace instance.
- **Telemetry Events ::** Provide insight into resource usage and provide data for anomaly detection.
- **Multiple Project Roots ::** Support for multiple project root directories within a single workspace (in case multiple projects are combined within the same workspace).
- **Additional Editor Support ::** Support for additional editors like Vim, Emacs, Sublime, Eclipse, and others.

References

[1] F1R3FLY.io. Rholang: Language Support for Rholang in Visual Studio Code, 2022. Available at: https://marketplace.visualstudio.com/items?itemName=F1R3FLY-io.f1r3fly-io-rholang.

[2] F1R3FLY.io. F1R3FLY Repository. GitHub, 2025. Available at: https://github.com/F1R3FLY-io/f1r3fly.

[3] F1R3FLY.io. Rholang Language Server Repository. GitHub, 2025. Available at: https://github.com/F1R3FLY-io/rholang-language-server.

[4] F1R3FLY.io. Rholang VSCode Client Repository. GitHub, 2025. Available at: https://github.com/F1R3FLY-io/rholang-vscode-client.

[5] Red Hat, Inc. Red Hat, Codenvy and Microsoft Collaborate on Language Server Protocol. Press Release, 2016. Available at: https://www.redhat.com/en/about/press-releases/red-hat-codenvy-and-microsoft-collaborate-language-server-protocol.

F1R3FLY.io, 9336 California Ave SW, Seattle, Washington 98136, USA

Email address: dylon@vinarytree.io