Open Source 101. Workshop.

Authoring Language Support Extension for the Zed Code Editor

Open Tech Summit Indonesia

November 26th, 2024

Alright, before we start,
I would like to introduce myself!

Wisnu Adi Nurcahyo

Software Engineer @ Traveloka

hi[at]wisn[dot]ch

github.com/wisn

Feel free to take a look at the Ada programming language extension that I authored on

github.com/wisn/zed-ada-language

Prerequisites

Make sure to have everything prepared!

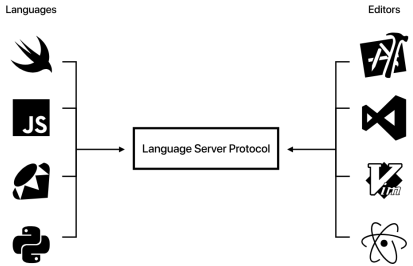
Prerequisites

- Use either Linux or macOS.
  - May use Windows if you are willing to build Zed from the source.
  - I personally use Fedora Linux.
  - This workshop is having an assumption that you are using Linux.
- Must have Zed code editor installed (latest version).
- Must have Git version control installed (latest version if possible).
- Must have Rust programming language installed (latest version if possible).

Outline

Find a tree-sitter grammar for the target language. This is used for the syntax highlighting feature.

Implement the language extension with Rust. Glue all of them together!



Find an LSP (language server protocol) implementation for the target language. Depending on the implementation, usually we will get code completion & jump to definition feature.

Publish the language extension by sending a pull request to github.com/zed-industries/extensions

Edit and preview the language extension directly on the Zed code editor!

Let’s code!

We will use the Ada programming language extension as our example.

Workspace Preparation

Prepare the code structure. It looks like this below.

```
otsi2024-workshop
├── Cargo.toml                <-- this is a Rust project so we need this
├── extension.toml            <-- the extension detail goes here
├── languages                 <-- describes the language (syntax highlighting etc)
│   └── ada                   <-- the name of the language
│       ├── config.toml       <-- config for the language (file extension etc)
│       └── highlights.scm    <-- grammar queries to support syntax highlighting
├── LICENSE                   <-- what is the license of the project?
├── README.md                 <-- self-explanatory
└── src                       <-- the source directory of the Rust project
    └── ada.rs                <-- describes the extension integration
                                  in this case, we add language server support
                                  (code completion, jump to definition, etc)
```

Describe the Project Detail

```
# Cargo.toml

[package]
name = "zed_ada"
version = "0.1.0"
edition = "2021"
publish = false
license = "MIT"

[lib]
path = "src/ada.rs"            <-- describes the entrypoint
crate-type = ["cdylib"]

[dependencies]
zed_extension_api = "0.2.0"    <-- adds Zed extension API as our dependency
```

Describe the Extension Detail

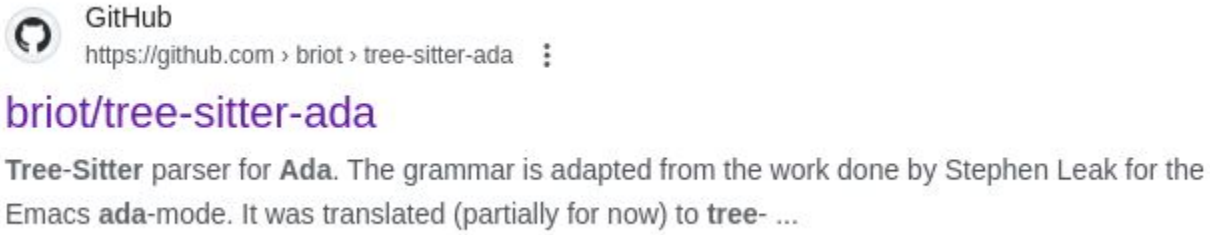
```toml
# extension.toml

id = "ada"
name = "Ada"
version = "0.1.0"
authors = ["wisn <hi@wisn.ch>"]
description = "Ada language support for Zed."
repository = "https://github.com/wisn/otsi2024-workshop"
schema_version = 1
```

First Step

Syntax highlighting support!

Find the Tree-sitter Grammar

Try to search for “tree sitter ada” on your favorite search engine.
One of the result should be the briot/tree-sitter-ada repository.



We are lucky that the tree-sitter grammar for Ada is available. Let’s use this grammar!
Oh, what if there is no tree-sitter grammar available for our target language?
Well, we need to create the grammar ourselves. See tree-sitter.github.io for the detail.

Describe the Tree-sitter Grammar

```toml
# extension.toml

id = "ada"
name = "Ada"
version = "0.1.0"
authors = ["wisn <hi@wisn.ch>"]
description = "Ada language support for Zed."
repository = "https://github.com/wisn/otsi2024-workshop"
schema_version = 1

# add the grammar config
[grammars.ada]
repository = "https://github.com/briot/tree-sitter-ada"
commit = "e8e2515465cc2d7c444498e68bdb9f1d86767f95"
```

Describe the Language Config

```
# languages/ada/config.toml

name = "Ada"
grammar = "ada"
path_suffixes = ["ads", "adb"]      <-- auto apply for these file extensions
line_comments = ["-- "]             <-- describe keyword for a single line comment
brackets = [                        <-- quality of life config
    { start = "(", end = ")", close = true, newline = true },
    { start = "[", end = "]", close = true, newline = true },
    { start = "{", end = "}", close = true, newline = true },
    { start = "<<", end = ">>", close = true, newline = false },
    { start = "\"", end = "\"", close = true, newline = false, not_in = [
        "string",
    ] },
    { start = "'", end = "'", close = true, newline = false, not_in = [
        "string",
    ] },
]
```

Put a Placeholder for the Language Server

```rust
// src/ada.rs

use zed_extension_api::{self as zed, EnvVars, LanguageServerId, Result};

struct AdaExtension;

impl zed::Extension for AdaExtension {
    fn new() -> Self {
        Self
    }

    fn language_server_command(
        &mut self,
        _language_server_id: &LanguageServerId,
        _worktree: &zed::Worktree,
    ) -> Result<zed::Command> {
        Ok(zed::Command {
            command: String::new(),
            args: vec![],
            env: EnvVars::new(),
        })
    }
}

zed::register_extension!(AdaExtension);
```

Querying and Labeling the Grammar

We will add these lines below just to test whether the syntax highlighting works.

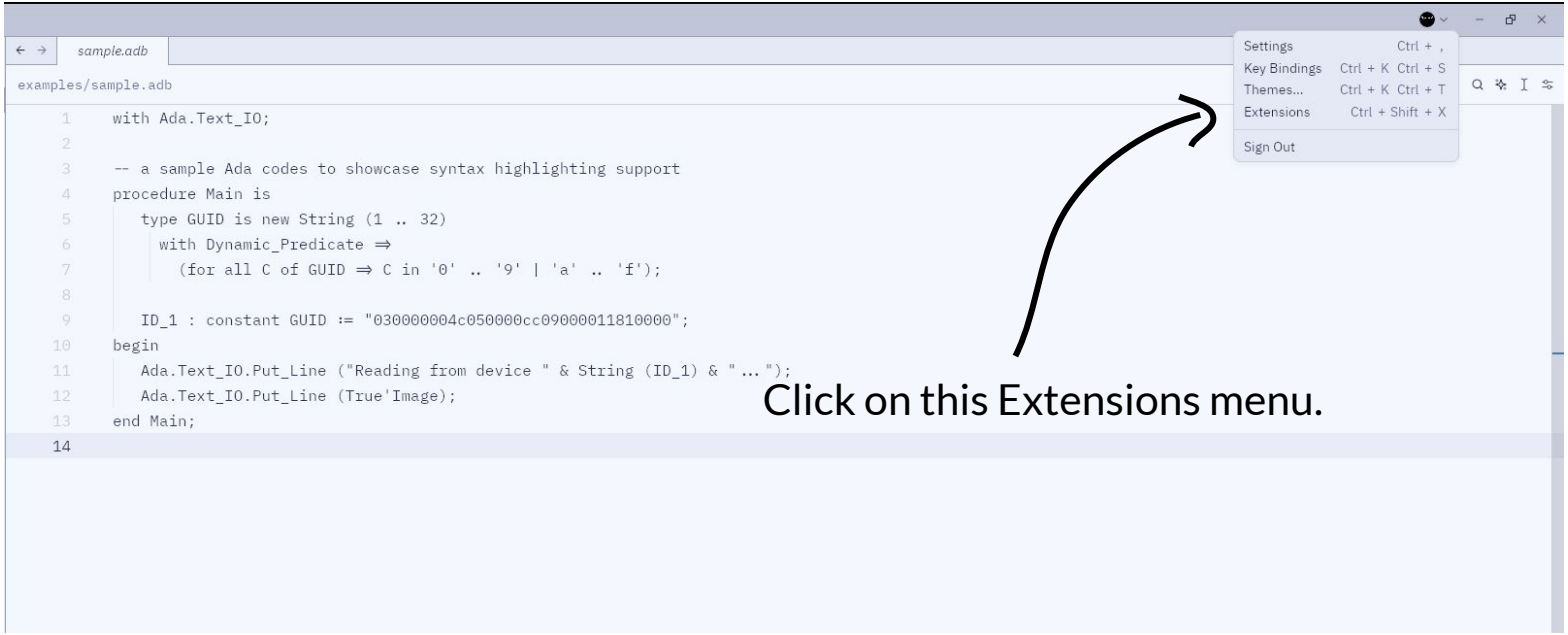
```
; languages/ada/highlights.scm

; comments
(comment) @comment

; literals
(string_literal) @string
(character_literal) @string
(numeric_literal) @number
((identifier) @boolean (#match? @boolean "^[Tt][Rr][Uu][Ee]$"))
((identifier) @boolean (#match? @boolean "^[Ff][Aa][Ll][Ss][Ee]$"))
```

Learn more about it on tree-sitter.github.io/tree-sitter/using-parsers#query-syntax

Create a Sample Ada File

```
-- examples/sample.adb

with Ada.Text_IO;

-- a sample Ada codes to showcase syntax highlighting support
procedure Main is
   type GUID is new String (1 .. 32)
     with Dynamic_Predicate =>
       (for all C of GUID => C in '0' .. '9' | 'a' .. 'f');

   ID_1 : constant GUID := "030000004c050000cc09000011810000";
begin
   Ada.Text_IO.Put_Line ("Reading from device " & String (ID_1) & "...");
   Ada.Text_IO.Put_Line (True'Image);
end Main;
```

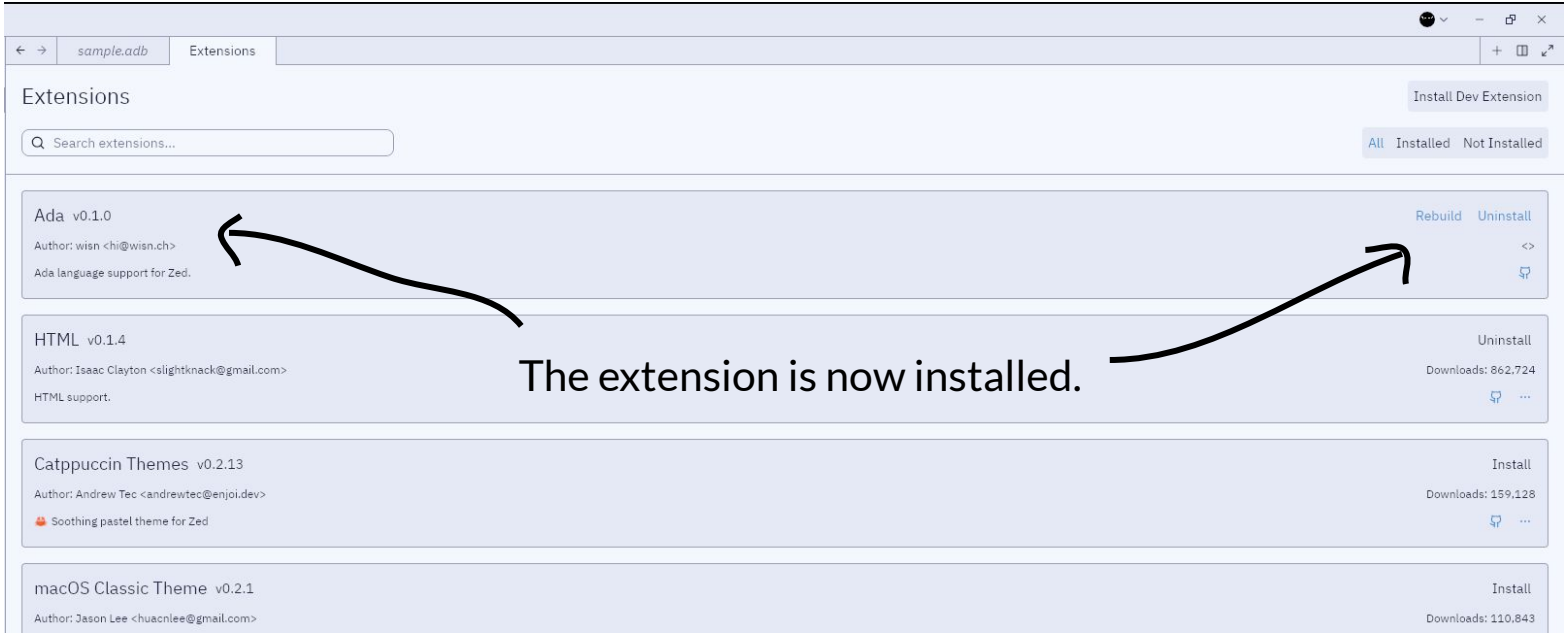
Install the Extension Locally



Install the Extension Locally

Then, choose the extension project root directory. Wait until the installation is finished.

Install the Extension Locally



The extension is now installed.

Open the Ada File and Check the Syntax Highlighting

sample.adb | Extensions

examples/sample.adb

```
with Ada.Text_IO;

-- a sample Ada codes to showcase syntax highlighting support
procedure Main is
   type GUID is new String (1 .. 32)
     with Dynamic_Predicate =>
       (for all C of GUID => C in '0' .. '9' | 'a' .. 'f');

   ID_1 : constant GUID := "030000004c050000cc09000011810000";
begin
   Ada.Text_IO.Put_Line ("Reading from device " & String (ID_1) & "...");
   Ada.Text_IO.Put_Line (True'Image);
end Main;
```

Notice that some of the codes are highlighted. It works! Now, time to add more!

Describe the Syntax Highlighting Query

Since the queries are too many to be typed and shown here, we can just copy and paste it from the workshop repository source.

github.com/wisn/otsi2024-workshop/blob/master/languages/ada/highlights.scm

Other than the comment and literals part that we have added before, just copy and paste the rest to our languages/ada/highlights.scm file.

Then, rebuild the extension.

The Final Looks of the Syntax Highlighting Support

sample.adb

examples/sample.adb

```
with Ada.Text_IO;

-- a sample Ada codes to showcase syntax highlighting support
procedure Main is
   type GUID is new String (1 .. 32)
     with Dynamic_Predicate =>
       (for all C of GUID => C in '0' .. '9' | 'a' .. 'f');

   ID_1 : constant GUID := "030000004c050000cc09000011810000";
begin
   Ada.Text_IO.Put_Line ("Reading from device " & String (ID_1) & "...");
   Ada.Text_IO.Put_Line (True'Image);
end Main;
```

Second Step

Language server integration!

Find the Language Server Implementation

Try to search for “language server ada” on your favorite search engine.
One of the top result should be AdaCore/ada_language_server repository.

We are lucky that the language server for Ada is available. We can use this!
What happens if there is no language server available?
Well, we need to create one ourselves which is not an easy task.
See langserver.org for more information (and known implementation).

Find a Reference for the Integration

Integrating a language server might not be easy and it need to be implemented using the Rust programming language.

Fortunately, if we can read some codes (and dare enough to edit it), we can just look for a reference and apply it to our case.

In this case, I use this implementation below as a reference.

github.com/zed-industries/zed/blob/main/extensions/zig/src/zig.rs

Then, modify it to my need.

Update the Extension Config

```toml
# extension.toml

id = "ada"
name = "Ada"
version = "0.1.0"
authors = ["wisn <hi@wisn.ch>"]
description = "Ada language support for Zed."
repository = "https://github.com/wisn/zed-ada-language"
schema_version = 1

[grammars.ada]
repository = "https://github.com/briot/tree-sitter-ada"
commit = "e8e2515465cc2d7c444498e68bdb9f1d86767f95"

[language_servers.als]
name = "Ada Language Server"
language = "Ada"
```

Implement the Language Server Integration

We need to type so many codes for this.

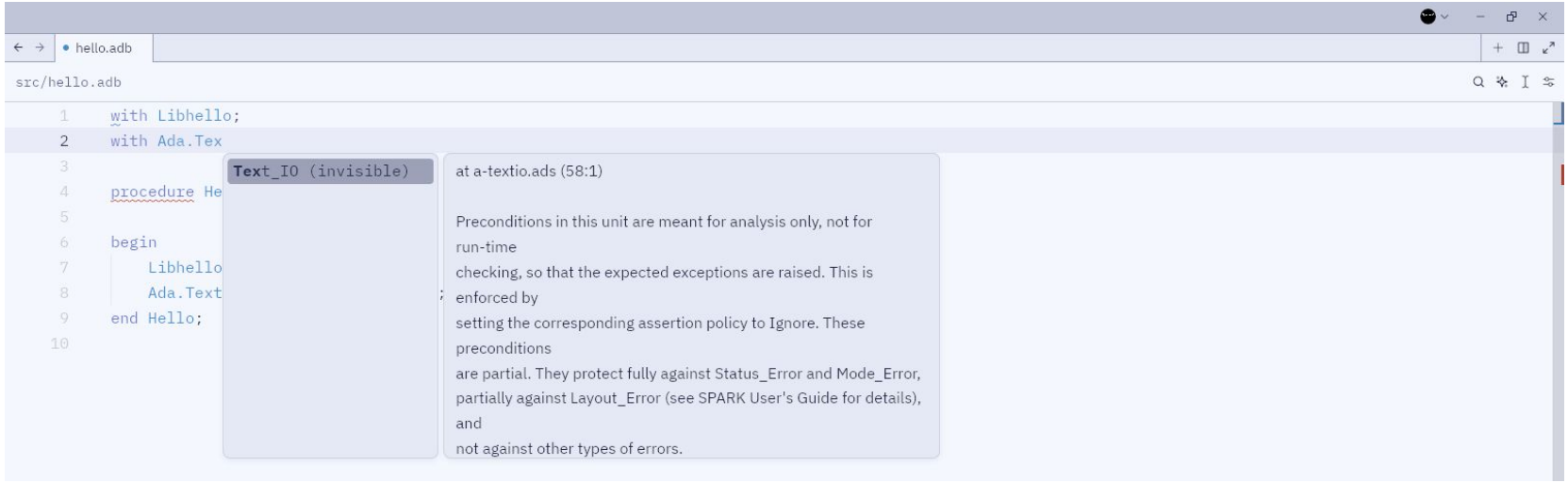
Thus, since the codes are too many to be shown here, please copy and paste the codes from this file below.

github.com/wisn/otsi2024-workshop/blob/master/src/ada.rs

I will try to explain the important parts.

Don’t forget to rebuild the extension!

Does the Language Server Works?



Yes, it works! We can now able to get some feedback from the language server.

What’s next?

Add the extension to Zed by sending a pull request to
github.com/zed-industries/extensions

Fin

Any questions?

Need the workshop resource? Feel free to visit this repository below!
github.com/wisn/otsi2024-workshop