Basic LSP Implementation

Author: Nikita Vetluzhskikh
Date: July 14, 2025
GitHub repository

Introduction

This project demonstrates a simple LSP implementation using standard I/O. It includes dictionary-based word completion and basic diagnostics, develop within one week.

1 Project Structure

The project consists of the following files and directories:

- CompilerHandler/ - Compiler processing
  - `Compiler.h` - main Class for compiler running
  - `CompilerOutput.h` - structrures for JSON like output
- `external/` — Third-party dependencies.
  - `json.hpp` — Single-header JSON parser (nlohmann/json).
  - `test-VSCode-external-base` — VSCode client used for testing the server.
- `Logger/` — Logging module.
  - `Logger.cpp` — Logger implementation.
  - `Logger.h` — Logger interface.
- `Messages/` — Message types and protocol logic.
  - `methods/` — Protocol-specific methods.
    - `InitializeResult.h` — Initialization result structure.
    - `ServerInfo.h` — Server metadata structure.
  - `TextDocument/` — Text document handling logic.
    - `Params/` — Request parameter structures (e.g., `CompletionParams.h`, `DidChangeTextParams.h`)
    - `Result/` — Completion and diagnostic structures
    - Other files: `TextDocument.cpp`, `TextDocument.h`, `TextDocumentItem.h`, `TextDocumentSync.h`, etc.
  - `DictionaryWords.*` — Dictionary lookup logic.
  - `Message.h` — Common message definitions.
  - `NotificationMessage.h, RequestMessage.h, ResponseMessage.h` — Message categories.
  - `ResponseResult.h` — Structure for result responses.
  - `Range.h` — Text range representation.
- `main.cpp` — Server application entry point.
- `CMakeLists.txt` — Build configuration.

2 Dependencies

- Language: C++
- OS: Ubuntu 24.04.2 LTS
- Libraries: STL, nlohmann/json (3.11.2)

3 How It Works

The client starts the Language Server Protocol and sends an `initialize` request. Upon receiving a response from the server with its capabilities, it sends an `initialized` notification, confirming the setup.

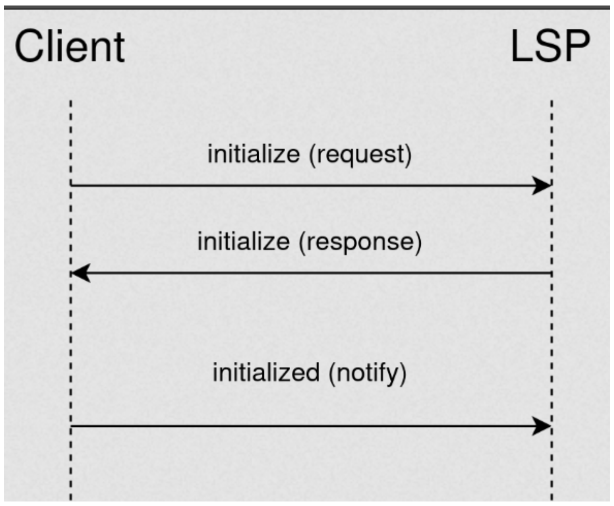


Figure 1: Initialization diagram

Communication uses standard input and output (`stdin` and `stdout`). `stderr` can be used for logging purposes.

When a file is edited, the client sends a `textDocument/completion` request (for word suggestions (At the moment work only with Keywords)) and a `textDocument/diagnostic` request (launch a compiler).

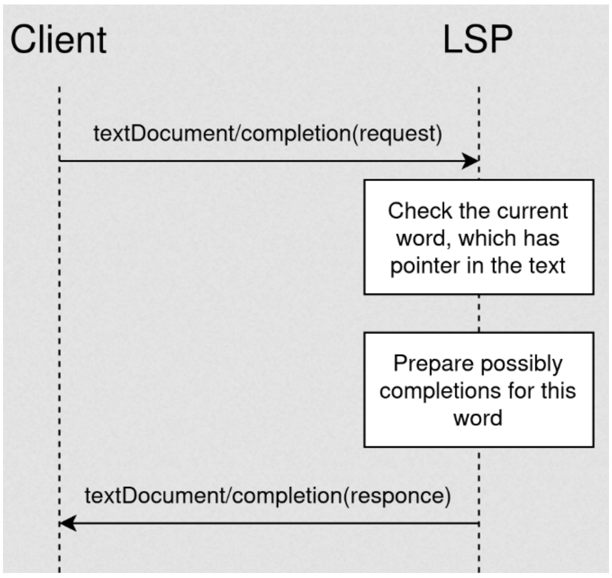

Client
LSP
textDocument/completion(request)
Check the current word, which has pointer in the text
Prepare possibly completions for this word
textDocument/completion(responce)

Figure 2: Completion workflow diagram

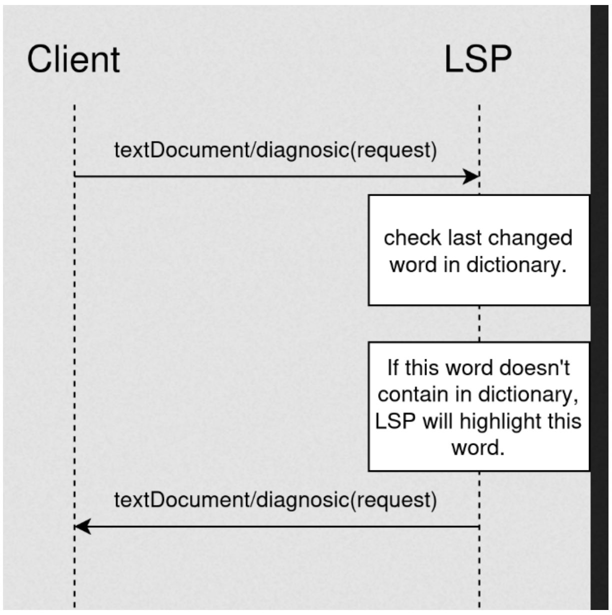
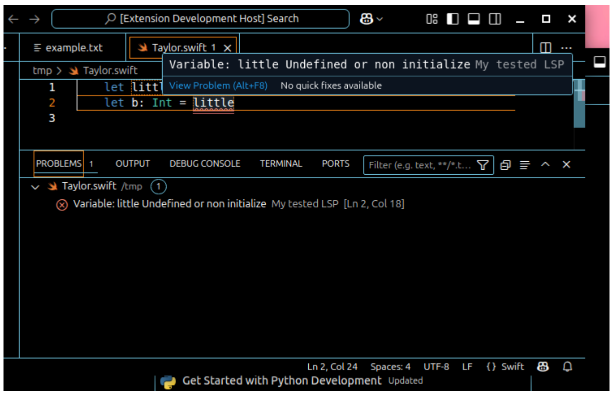

Client
LSP
textDocument/diagnosic(request)
check last changed word in dictionary.
If this word doesn't contain in dictionary, LSP will highlight this word.
textDocument/diagnosic(request)

Figure 3: Diagnostic workflow diagram



```
let littl
let b: Int = little
```

```
Variable: little Undefined or non initialize My tested LSP
```

```
Variable: little Undefined or non initialize My tested LSP [Ln 2, Col 18]
```

Figure 4: Highlighting showcase

When user opens a new file, VSCode sends `textDocument/didOpen` notification, and the server saves this document filepath to map – filepath is key, last changes is value.

Notification `textDocument/didChanges` updates the value in the map.

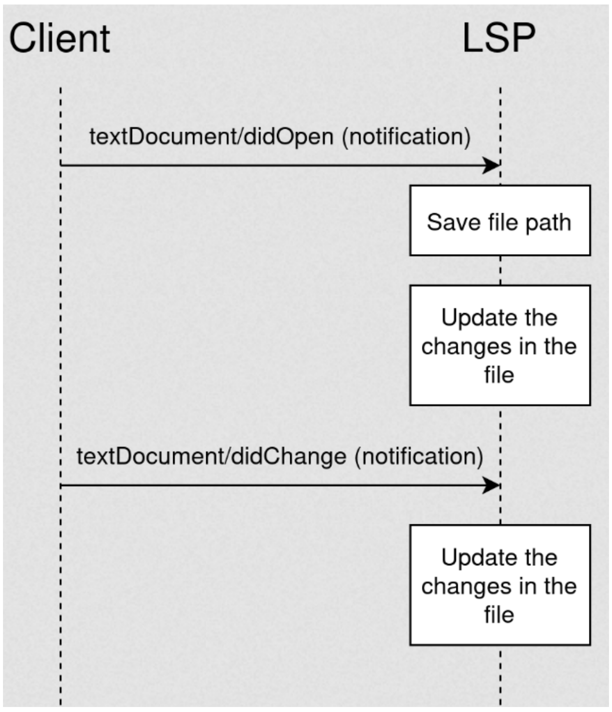

Client
LSP
textDocument/didOpen (notification)
Save file path
Update the changes in the file
textDocument/didChange (notification)
Update the changes in the file

Figure 5: textDocument/didOpen and didChanges

4 Language syntax

Language for my compiler, it's a sublanguage from swift. This language has a basic language structures.

```
// comments
// variable declaration
var a
var b: Int
var c: Double = 2.0
// variable declaration with expression
var d = (2 + 4) * 3.5
// const declaration
let a

// while cycle
while ( d > c) {
write("d more than c")
}

// conditions
if ( 5 <= 10 ){
    write("5 less than 10")
}
else {
 write("Is 5 less than 10? Something go wrong..."
}

// function creating
func foo(){
    return
}
// function with arguments
func foo(var_name some_id: Int){
    return
}
//function with returning type
func boo() -> Int {
    return 2
}
// function call
write("Hello world)

// internal functions
// more in the compiler's documentation
```

5 Compiler processing

The server runs the compiler when the textDocument/diagnostic function is called. If an error occurs, the compiler returns a message in JSON format. The compilation is terminated when the first error is detected.

Details about the compiler implementation can be found in its documentation prepared for the project defence.

The following error types are provided in my implementation of the compiler:

```
    enum ErrorCode{
        ER_NONE          = 0,
        ER_LEX           = 1, // Lexical error when the token cannot be identified
        ER_SYNTAX        = 2, // Syntax error when token order is violated
        ER_UNDEF_FUNC_OR_REDEF_VAR = 3,
        ER_PARAMS        = 4,   // Semantic error in the program - wrong number
                                // type of parameters in the function call
                                // or wrong type of return value from
                                // the function
        ER_UNDEF_VAR_OR_NOTINIT_VAR = 5, // Semantic error in the program - using an
                                        // undefined or uninitialized variable
        ER_FUNC_RETURN   = 6, // Semantic error in the program - missing/absent
                            // expression in the function return statement
        ER_TYPE_COMP     = 7, // Semantic type compatibility error in arithmetic string
                            // and relational expressions
        ER_INFERENCE     = 8, // Semantic type derivation error - the type of the
                            // variable or parameter is not specified and cannot be
                            // derived from the expression used
        ER_OTHER_SEM       = 9, // Other semantic errors
        ER_PARAMS_ARGS_MISMATCH = 40, // Concretisation of ER_PARAMS error for the server
        ER_PARAMS_TYPE_MISMATCH = 41, // Concretisation of ER_PARAMS error for the server
        ER_INTERNAL      = 99 // Internal compiler error i.e. not influenced by the
                            //input program
    };
```

JSON variables containing additional information are provided for all errors. At the moment, not all errors contain exact data - such errors are marked with the message **"Unresolved error"**.

The server can handle errors of types `ER_UNDEF_OR_NON_INIT_VAR` and `ER_SYNTAX`.

5.1 ER_LEX

The compiler returns the following data: ...

```
{ // json message for ER_LEX
        "error_code": 1,
        "message": "Lexical error when check this: 1.2",
        "line": 1,
        "char_pos": 5
}
```

```
// the code which throws error ER_SYNTAX
1.2.3
```

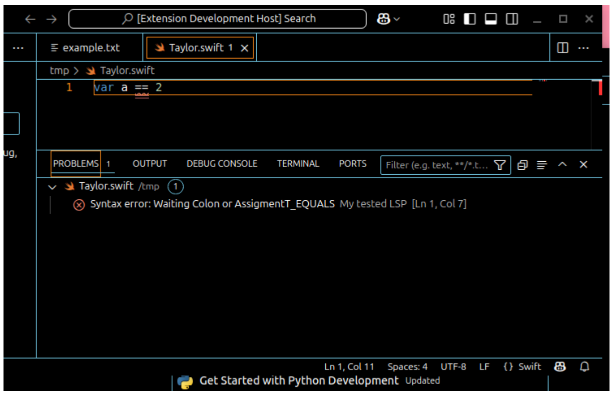
Nothing too complicated. There is no parsing algorithm on the server side. It only outputs the result based on JSON message

5.2 ER_SYNTAX

If the compiler fails to match the input text with LL rules, it raises a syntax error. In this case, it reports which token was received and which was expected (sometimes only the text of the error message is returned).

```
{ // json message for ER_SYNTAX
        "error_code": 2,
        "message": "Syntax error: Waiting keyword or variable doesnt have nil_flag",
        "line": 2,
        "char_pos": 10,
        "token_type":33, // enum with type of token
        "token_string":"null" // if the compiler can find out the token identifier
}
```

```
// the code which throws error ER_UNDEF_VAR_OR_NOT_INIT_VAR
let a: Int
let b: Int = a
```



```
var a == 2
```

```
Taylor.swift /tmp 1
Syntax error: Waiting Colon or AssigmentT_EQUALS My tested LSP [Ln 1, Col 7]
```

Figure 6: ER_SYNTAX showcase

5.3 ER_UNDEF_OR_NON_INIT_VAR

The compiler returns the following data: ...

```
{ // json message for ER_SYNTAX
        "error_code": 2,
        "message": "Syntax error: Waiting keyword or variable doesnt have nil_flag",
        "line": 2,
        "char_pos": 10,
        "token_type":33, // enum with type of token
        "token_string":"null" // if the compiler can find out the token identifier
}
```

```
// the code which throws error ER_SYNTAX
var abc:
```

Thanks to the already detected variable name, it is sufficient to locate this variable in the specified line and indicate its position to the text editor.

Thanks to the already found variable name, you only need to find this variable in the specified line and pass the text editor the exact position of the variable in the text.

5.4 Runtime error

In some cases, the compiler would go into infinite recursion. If this happens, the server waits 10 seconds, resets the compiler and sends an error.

```
// code with infinite recursion for example
Double

var a :Int
```

6 Implemented Features

6.1 initialize

Establishes initial communication. Returns `ServerCapabilities` and `ServerInfo`.

```
{ // responce from server
    "id": 0,
    "jsonrpc": "2.0",
    "result": {
        "capabilities": {
        "completionProvider": {},
        "diagnosticProvider": {
            "interFileDependencies": false,
            "workspaceDiagnostics": false
        },
          "textDocumentSync": 1
        },
        "serverInfo": {
          "name": "my-lsp-server",
          "version": "0.0.1"
        }
    }
}
```

6.2 textDocument/completion

Returns word suggestions. If results exceed 1000 items, the server sets `isIncomplete` to `true`.

```
{ //request from client
"id":14,
"jsonrpc":"2.0",
"method":"textDocument/completion",
"params":{
    "context":{
        "triggerKind":3
    },
    "position":{
        "character":8,
        "line":2
    },
    "textDocument":{
        "uri":"file:///tmp/example.txt"
    }
}}
```

```
{ //responce from client
"id":14,
"jsonrpc":"2.0",
"result":{
    "isIncomplete":false, //count of items less than 1000 -> false
    "items":[
        {"label":"truths"},
        {"label":"truth's"},
        {"label":"truthiness"},
        // many words
        {"label":"truckled"},
        {"label":"truncating"}
    ]
    }
}
```

6.3 textDocument/didOpen

Notifies the server that a file was opened. The server stores the file path.

```
{ // notification from client
"jsonrpc":"2.0",
"method":"textDocument/didOpen",
"params":{
    "textDocument":{
        "languageId":"plaintext",
        "text":"hello\n\ntest \n\nthis the first file\n\nthis the fisst file",
        "uri":"file:///tmp/example.txt",
        "version":1
        }
    }
}
```

6.4 textDocument/didChange

Notifies the server of changes on an opened file.

```
{ // notification from client
"jsonrpc":"2.0",
"method":"textDocument/didChange",
"params":{
    "contentChanges":[
        {
            "text":"hello\n\ntest trust\n\nthis the first file\n\nthis the fisst file"
        }
    ],
    "textDocument":{
        "uri":"file:///tmp/example.txt",
        "version":12
    }
}}
```

6.5 textDocument/diagnostic

Client requests file diagnostics. The server analyzes text and returns any found issues.

```
{ // responce from server
"id":1,
"jsonrpc":"2.0",
"result":{
    "items":
        [{
            "message":"fisst is not in our dictionary",
            "range":{
                "end":{
                    "character":14,
                    "line":6
                },
                "start":{
                "character":9,
                "line":6
                }
            },
            "severity":1,
            "source":"My tested LSP"
        }],
        "kind":"full"
    }
}
```

6.6 shutdown

Client send shutdown, and the server stops responding to any messages from the client except initialise or exit

6.7 exit

The server finish the process

7 Install

Run

```
git clone --recurse-submodule https://github.com/SyruSTR/test_lsp.git {DIRECTORY_PATH}
```

8 Running the Project

Run `start.sh` to launch VSCode and start the client. Press Ctrl+Shift+D, will open window 'Run and Debug'. Choose **Launch Client (Release Server)**

The extension will start in a new VSCode window. To run it, we need to create a file with a **.swift** extension. You can select the language manually, but LSP does not support working with raw text. It needs the file (It doesn't need to be saved). Initialisation will take place, but the client will not send requests to the server.

There can be several files open at the same time. But the compiler was not originally intended to work with several files. It means that the compiler will check each file separately and you cannot use variables and functions from another file.

9 Debugging

For debugging I use CMAKE profile with added flag **-DDEBUGGING=ON**. Next, I build my project with this flag. Run the extension in VSCode with profile **Launch Client (Debug Server)**. The last step, press Ctrl+Alt+5 in CLion (it's activating **'Attach to a process'**). And select `my-lsp-server`.

I can put breakpoints in the code or check logs