

echo $(whoami)

- Principle Software Engineer
- Co-founder and Chief Terminal Nerd @ PHL Code Club

Here is me with me cat, Fritzy =3



Some Helpful Acronyms

- LSP: Language Server Protocol
- IPC: Inter Process Communication
- JSON: JavaScript Object Notation
- RPC: Remote Procedure Call
- STDIO: Standard Input/Output

Setting Expectations

What this talk is:

- An introduction to LSP and related concepts like IPC and JSON RPC
- A brief look at some of the cool features that LSP provides
- Some time to look at cool technology

What this isn't

- A full technical deep dive into LSP
- An authoritative list of the capabilities of LSP
- An endorsement of Microsoft, ew

Overview

The Five W's of LSP[^1]

Basics of IPC

Defining JSON RPC

Language Servers and Clients

Common Messages

Negotiating Capabilities

Sample Flow

`tsserver`, the Typescript Language Server

Who is LSP

LSP was initially designed at Microsoft to power their new text editor.
You may have heard of it... VSCode!

When is LSP

The initial commit for LSP was in 2015 when Microsoft started work on VSCode with it being open sourced in 2016.

Powering tooling in editors such as VSCode, Neovim, Zed and more for years!

What is LSP

Language Server Protocol, or LSP, is a specification that aims to standardize communication between Language Servers and Clients. More on those later

Where is LSP

The specification is **Open Source** (*https://microsoft.github.io/language-server-protocol/*).

It likely powers the editor you use everyday, as well as other tools like:

- In browser editors like Monaco and CodeMirror
- Inside AI Agents like OpenCode and Goose
- The Symbols tab on GitHub (That's technically LSIF, the sister to LSP)

Why is LSP

Each tool needs custom code to handle each language for autocomplete, symbols, refactors, etc.

With LSP each tool has one LSP client (Generally) and each language has one Language Server (Hopefully).

Why is LSP: Table

✖	Go	Rust	Gleam	Typescript
VSCode	✅	✅	✅	✅
Neovim	✅	✅	✅	✅
Zed	✅	✅	✅	✅

Language	
Go	✅
Rust	✅
Gleam	✅
Typescript	✅

+

Editor	
Neovim	✅
VSCode	✅
Zed	✅

Basics of IPC

IPC is how one piece of software talks to another, usually on the same machine.

LSP generally uses STDIO for communication. It is comprised of:

- stdin
- stdout
- stderr

Defining JSON RPC

JSON RPC is a simple protocol for using JSON messages to handle remote procedure calls. It is transport agnostic, so you can have JSON RPC API's over HTTP as well as something like STDIO.

There are 4 main object types:

- Requests
- Responses
- Client Notifications
- Server Notifications

LSP Components

The LSP spec defines 2 distinct components:

- Clients
  - How you interact with a Language Server. Takes LSP responses/notifications and can perform actions on them.
- Language Servers
  - Handles language specific details. Computes diagnostics, keeps track of symbols, and can offer refactors (sometimes called Code Actions)

Language Servers

A process that gets started and owned by the Client to handle requests and emit notifications

Where all the language specific code lives

Decoupled from the details of the Client

LSP Clients

Spawns and manages Language Server processes and communication with them using IPC (Coming up next)

Takes actions inside of the source software based on responses and notifications.

Generic for all Language Servers (*hopefully*)

Protocol Message Structure

The spec divides messages into two parts separated by \r\n:

- Header part
  - Metadata about the message
- Content part
  - The actual data of the message in JSON form.

Message Headers

Two spec compliant headers:

- Content-Length
  - Required
  - The length of the content part in bytes
  - example: `Content-Length: 16\r\n`
- Content-Type
  - Optional
  - The mime type of the content part
  - defaults to `application/vscode-jsonrpc;charset=utf-8`
  - example: `Content-Type: application/json\r\n`

Message Content

Example JSON RPC Message:

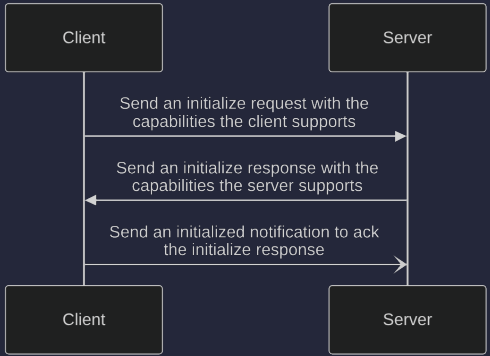
```
Content-Length: 118\r\n
\r\n
{
  "jsonrpc": "2.0",
  "id": "1",
  "method": "namespace/methodName",
  "params": {
    "someParam": "PhillyJS"
  }
}
```

Common Messages

Some common features that you might not know are powered by LSP.

- Autocomplete
- Go to definition/declaration/definition/implementation
- Find References
- Code Lenses/Code Actions
- Diagnostics

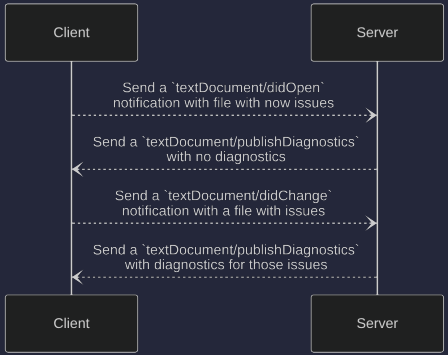
Negotiating Capabilities



Document Synchronization

- `textDocument/didOpen`
  - Sent when you first open a document to notify the language server so it can do things like compute diagnostics
- `textDocument/didChange`
  - Sent when you you make an edit to an existing document so the language server can *recompute* things like diagnostics
- `textDocument/didClose`
  - Sent when you close a document so the language server can cleanup any resources tied to that document

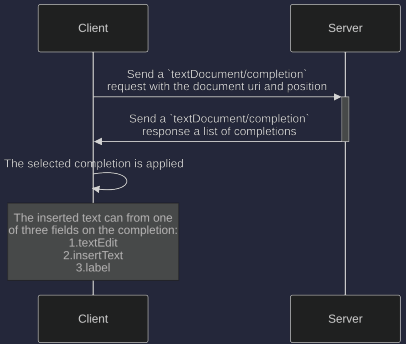
Document Synchronization Example



Sample Message: Diagnostics

```
{
    "method": "textDocument/publishDiagnostics",
    "params": {
      "uri": "file:///path/to/file.ts",
      "diagnostics": [
        {
          "range": {...},
          "severity": 1,
          "message": "BROKEN :("
        }
      ]
    }
}
```

Sample Flow: Autocomplete - Diagram



Sample Flow: Autocomplete - Request

```
{
    "jsonrpc": "2.0",
    "id": 42,
    "method": "textDocument/completion",
    "params": {
      "textDocument": {
        "uri": "file:///path/to/file.ts"
      },
      "position": {
          "line": 19,
          "character": 93
      }
    }
}
```

Sample Flow: Autocomplete - Response

```
{
    "jsonrpc": "2.0",
    "id": 42,
    "result": [
      {
        "label": "Text Suggestion",
        "kind": 1,
      },
      // ...other items
    ]
}
```

Basic LSP Client

I had initially planned on digging into the TypeScript language server code, but that stuff is **dense**. So instead I put together a little example of a client that we can use to interact with a language server and see what the communication looks like!

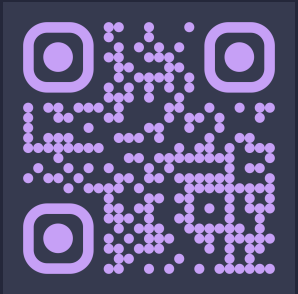
With Microsoft again putting a wrench in my plan, the Typescript LSP is... *weird*. So we will be looking at the Go language server: `gopls`!

Citations

1. **TJ Devries LSP Video** (*https://www.youtube.com/watch?v=LaS32vctfOY*)

$(which gvasquez)

My website! (*https://gvasquez.dev/*)



PHL Code Club (*https://phlcode.club/*)

