Developing a Visual Studio Code Language Support Extension for the Snail Programming Language

Charles Reinhardt

June 12, 2023

Abstract

A high quality programming environment (often an integrated development environment, or IDE) can be vital to enhancing developer productivity. Visual Studio Code (VS Code) is a popular, open-source text editor maintained by Microsoft. VS Code delivers language-specific features through freely dowloadable, community-built extensions on an online marketplace. Many of these extensions allow developers to take advantage of editing features such as syntax highlighting, code-autocompletion, or debugging support. The snail language (Strings Numbers Arrays and Inheritance Language) is a simple, object-oriented programming langauge meant to be implemented in a one-semester undergraduate course. We present a new VS Code extension to provide language support for the snail language. The extension implements support for syntax highlighting, rudimentary auto-completion, and static error-checking diagnostics using VS Code's Langauge Server Protocol. This report summarizes the contents of a VS Code extension and gives an overview of how an extension runs, particularly highlighting the functions of VS Code's Langauge Server Protocol. I also discuss how this extension can be futher developed to make use of VS Code's Debug Adapter Protocol to implement a debugger with breakpoints, start/stop behavior, and variable inspection.

1 Introduction

Much of software development in the present day takes place in integrated development environments (IDEs) [20]. An IDE is a collection of software development tools, such as a code editor, debugger, and build system, often unified under a similar user interface, with te goal of simplifying the software development process and enhancing developer productivity [14, 40]. In addition to composing these tools together, many IDEs also offer advanced features within their code editors such as syntax highlighting, code auto-completion, and error-checking diagnostics. Today, there are any number of different IDEs available for use with any given programming language, many offered as freely-downloadable for public use. For example, a developer looking to write code in Java may choose to do so in Eclipse, IntelliJ IDEA, or BlueJ [11, 17, 21]. With so many high quality IDEs available today, it is no surprise that 75% of software developers today use an IDE in their everyday work [20]. Clearly, the IDE has become an integral part of the software development process today [43].

Visual Studio Code (VS Code), is a popular, open source text editor maintained by Microsoft [31, 36]. When first downloaded, VS Code is a lightweight text editor with minimal features. However, a number of community-built, freely downloadable extensions offer advanced langauge features. These extensions can be dowloaded on VS Code's online extension marketplace, and can turn VS Code into a very fast ,robust, and powerful development environment for any programming task or language [33].

The snail programming language (Strings Numbers Arrays and Inheritance Language) is a simple, object-oriented programming language meant to be implemented in a one-semester undergraduate course [3]. In order to be implemented in a short time frame, snail is defined by limited features and a relatively annoying syntax. The language lacks a for-loop structure, opting instead to provide only while-loops. Each if statment *requires* an else clause, even when the developer does not need to use one. Every statement must end with a semi-colon, which is not an issue until you accidentally forget one, and the resulting parse error message is wildly uninformative. While this design makes snail easier to implement, it makes it hard for a developer to write programs in snail.

Currently, there are no tools to offer advanced langauge support for the snail language. This is no surprise, as snail has a small user base and was first released in February 2022 [2]. With no external support for the language, software developers are taken out of their comfort zone and are offered no guidance when navigating the snail language.

This report presents the Snail Language Support VS Code extension, which seeks to address the lack of programming support tools for the snail programming language. Snail Langauge Support provides several important features to make programming in the snail language easier. First, it features syntax highlighting to make reading snail code easier and help highlight key structures or keywords of the language. Further, it features rudimentary auto completion with auto-closing brackets, braces, and quotes, as well as if-else, while loop, and class definition snippets, reducing the burden of memorizing snail's strict and unintuitive syntax. The extension also has automatic, real-time error checking diagnostics that allow a user to see syntax or parse errors in a snail program before running it for themselves. Finally, the extension is structured to support a full debugger with breakpoints, step-in, step-over, and step-out functionality.

This paper will outline the process of building the Snail Language Support VS Code extension. Specificaly, we will introduce the structure of a VS Code extension that is meant to add support for new programming langauges. We will also discuss how this extension uses VS Code's language server protocol (LSP) to provide realtime error diagnostics [30]. We will address how the Snail Language Support extension may be further developed to include debugging support with breakpoints, step in/out behavior, and variable inspection, particularly highlighting the rolw of VS Code's debug adapter protocol DAP [29]. Finally, we will review good software development practices such as version control and documentation.

2 Background

In this section, we will discuss the history of integrated development environments (IDEs) and debuggers, and how they both assist software developers today. We will also discuss Visual Studio Code (VS Code) in more detail, identifying the technologies that power VS Code.

2.1 History of the Modern Integrated Development Environment

The first programming environment to remotely resemble a modern IDE was the Dartmouth BASIC programming language run on the Dartmouth Time Sharing System (DTSS), developed in the mid 1960s [23, 24]. Dartmouth BASIC was an example of a compile and go system, where program compilation was not separated from program execution [46]. Additionally, the DTSS also placed focus on making sharing time on a single university computer a simpler task, in order to help make programming more accessible to novices. This makes the DTSS an early example of combining multiple software development tasks into a single programming environment.

Borland's TurboPascal takes this idea one step further. Released in 1983, TurboPascal featured a Pascal compiler, code editor, file navigation user interface, and a rudimentary debugger [12, 19, 45]. This time period would also see language-specific IDEs in Microsoft's Visual BASIC 1.0, Microsoft's QuickC, and Borland's Turbo C/C++, all featuring similar characteristics of early IDEs [8, 10, 37, 38].

While early IDEs certainly had their merits, they usually only supported one programming language. Microsoft's Visual Studio, released in 1997, was one of the first IDEs to package support for a variety of programming languages in one piece of software [28]. Visual Studio also featured extensive tools to aid in development for software on the early internet. T oday, we see a trend towards open source IDEs such as Eclipse, NetBeans, or VS Code [4, 11, 31]. Even open source editors such as these include advanced development features such as an intelligent code editor, debugging tools, and version control integration. Many of these editors have advanced features within their code editors, providing functions such syntax highlighting, code autocompletion, and realtime error diagnostics.

There are many benefits to developing software using modern IDEs. By bundling code editing, build systems, and program execution into one tool, modern IDEs reduce the time and effort a developer needs to put forth in order to test a piece of code [14]. This also reduces the number of decisions a developer has to make while developing code, which can help increase productivity. Using an IDE can also standardize the software development process, by either helping a group of people use a consistent UI (and know how to help eachother), or allow a developer to avoid switching applications to complete a single task [44].

2.2 History of Modern Debugging Tools

Debugging is the process of searching for and fixing unexpected errors in a piece of code [7]. Early techniques of debugging software involved physically printing machine output and reading, step-by-step, through code and output, searching for a potential error [39]. Some computer systems, such as the IBM 704 Data Processing system released in 2968, allowed programmers to print information stored in specified memory locations in specified forms to assist with debugging activities [5,6].

The adb and dbx debuggers for the Unix operating system saw a release in the 1970s and 1980s [25,26]. These tols introduced the concept of breakpoints, which help a developer pinpoint the location of a fault in a piece of code. Both adb and dbx were run on the command line. Next, came the GNU project debugger (gdb), another command line debugger, released in 1986, which gave the user even greater control of tracing a program's execution throughout its runtime [16]. With the ability to debug low-level languages like C and Assembly, gdb is still in use today.

Modern debuggers with graphical user interfaces (GUIs) include the Visual Studio Debugger, which allows a developer to track variable values, function calls, and even change pieces of code or values of expressions while debugging [22]. Eclipse, a popular IDE for Java, has a similar debugger with a user friendly GUI [41]. Today, many developement workflows include the use of debugging tools [27].

Debugging tools can be of great assistance to developers. By allowing developers to more closely and quickly inspect a program's execution, debugging tools can reduce the amount of time a developer spends debugging, and thus enhance developer productivity [48].

2.3 What is Visual Studio Code?

Visual Studio Code is a popular, open source code editor. Visual Studio Code is designed to be fast and lightweight, with a focus on allowing a developer to write, test, adn debug source code quickly [35]. To achieve this, VS Code uses native, web, and language-specific technologoies. Tools like Electron allow VS Code to run on multiple platforms and operating systems using common web technologies like JavaScript, HTML, and CSS [9].

While VS Code is lightweight and fast, it still supports advanced features such as build or debugger tools available through the online VS Code extension marketplace [33]. VS Code is designed for extensibility, providing robust APIs to allow independent developers to create and publish extensions [32]. As a result, VS Code is able to grow and develop along with its community of users.

Existing extensions make VS Code a popular code editor [36]. The CodeSnap extension allows a user to capture and share improved screenshots of code [1]. GitLens adds additional visualization tools to help developers contribute to Git repositories while editing in VS Code [15]. Language Support for Java(TM) provides intellisense, formatting, refactoring, and build system support for the Java language [18]. The result of this extension support is an IDE that is used by over 75% of professional and hobby software developers [36].

3 Anatomy of a Visual Studio Code Extension

TODO add signposting

3.1 Directory Structure

Most VS Code extensions share a similar directory structure. These structures may vary depending on what purpose the extension is meant to serve. In figure 3.1, we see a diagram of a typical language support VS Code extension. First, the .vscode directory configures how we run and test our exstension locally. Through the launch.json and tasks.json files, we define various tasks and configurations to streamline the build and testing process.[1] It is important to note that these configuration files do not get packaged with a released VS Code extension. Next, a README.md file that allows us to advertise the features of our extension. This file is rendered in the VS Code extension marketplace. We also have a tsconfig.json file, which defines our TypeScript code gets transpiled to JavaScript, a topic we will discuss in further detail later in this paper.

The src directory contains the source code that runs our extension. In this directory, the extension.ts file defines special actions for our extension and launches the extension within an existing VS Code window. The server.ts file defines a language server that allows our extension to take advantage of realtime error diagnostics. We further discuss language servers later in this paper.

Lastly, the package.json file defines which features and configurations our extension supports. It also contains some biographical information about the extension, such as the author, publisher, or name of the exstension.

To build the structure for Snail Language Support, we used the Yeoman tool to generate an extension skeleton [47]. This skeleton included configuration and source files for us to modify and build Snail Language Support on.

We build Snail Langauge Support with TypeScript. TypeScript is a syntactic superset of JavaScript that allows the developer the benefits of a type system without sacrificing JavaScript's flexibility [42]. Using the TypeScript compiler (tsc), a program in TypeScript is transpiled to an equivalent JavaScript program. With a tsconfig.json file, we can define how the TypeScript compiler handles our TypeScript code. We define where the compiler looks for our .ts files, where to place resulting JavaScript files, and the strictness of type checks during transpilation.

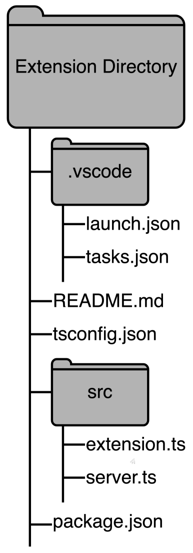

Figure 1: The typical directory structure of a VSCode language support extension

3.2 Extension Contributions: The Extension Manifest

Recall that the package.json defines what our extension contributes to the user. We now dive into what some of those contributions might be.

First, we can define some metadata about our extension, such as a name, author, description, and version. We can also define which version of the VS Code engine this extension expects. This ensures that an extension knows which VS Code APIs it has proper access to. For example, an extension that expects a VS Code version of 1.7.0 won't be able to take advantage of an API released in 1.8.0 [34]. We can also define some categories that our extension falls under, helping our extension gain visibility on the online VS Code marketplace.

[1]From personal experience, we strongly recommend taking the time to understand these files before diving into developing a VS Code extension. It will likely save painstaking debugging in the near future. See launch configurations: https://code.visualstudio.com/docs/editor/debugging#_launch-configurations and tasks: https://code.visualstudio.com/docs/editor/tasks

Next, and very importantly, we can define activations events that define when our extension becomes active. When VS Code detects a given activation event, it calls the activate function found in our extension.ts file. For example, onLanguage:snail says that Snail Language Support should be activated when a user is viewing a snail language file. We also define where VS Code looks for the activate function with our main attribute. For a more detailed look at Snail Language Support's package.json file, see Appendix A.

3.3 Extension Lifecycle: From Startup to Shutdown

Here, we discuss what happens under the hood when VS Code runs an extension.

When a VS Code client detects an extension's defined activation events, it reads the main attribute of package.json to call the activate function defined in that file. This file (in Snail Language Support's case, the extension.ts file) has access to the VS Code extension API, which allows us to update the VS Code client UI [32]. For example, depending on a user's configuration settings, Snail Language Support can display a VS Code UI style error message window to the user.

Also in this file, we register a few command handlers to watch for commands input from a user. These command handlers are examples of the observer design pattern, which allow a developer to handle a variety of input events that are experienced during runtime [13]. A developer can register an observer that waits and listens for a particular event, and executes a block of code once that event is detected. One of these command handlers responds to snail file debugging requests. We will touch on the topic of debugging within Snail Language Support more later.

We also launch our language server, which communicates via VS Code's Language Server Protocol (LSP), inside of our activate function. We further discuss this topic later.

3.4 Language Configuration

3.5 Rudimentary Autocompletion Through Snippets

3.6 Syntax Highlighting

4 Language Servers and the Language Server Protocol (LSP)

4.1 Theory

4.2 LSP in Snail Language Support

5 Debug Adapter Protocol

5.1 Theory

5.2 Debug Adapter Protocol in Snail Language Support

6 Good Practices in Software Development

6.1 Version Control

6.2 Documentation

7 Conclusions

References

[1] adpyke. Codesnap, Feb 2021.

[2] Kevin Angstadt. Version 1.0.0 released, February 2022.

[3] Kevin Angstadt. Introduction, May 2023.

[4] Apache. Apache netbeans releases, May 2023.

[5] IBM Archives, 2023.

[6] D. Arden, S. Best, F. Corbato, F. Helwig, J. McCarthy, A. Siegel, F. Verzuh, M. Watkins, and M. Weinstein. *Coding for the MIT-IBM 704 Computer*. Massachusetts Institute of Technology, 1957.

[7] AWS. What is debugging?, 2023.

[8] Michael Burgwin. History of development: Visual basic 1.0, May 2013.

[9] Electron. Build cross-platform desktop apps with javascript, html, and cs, 2023.

[10] Douglas Forer and Nicholas Petreley. Quickc is a one-stop development tool. *InfoWorld*, 13(46):113–114, Nov 1991.

[11] Eclipse Foundation. Eclipse installer 2023-03 r, Mar 2023.

[12] Zarko Gajic. Delphy history from pascal to embarcadero delphi xe 2, Mar 2017.

[13] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Observer*, page 293–303. Addison-Wesley, 1 edition, 1995.

[14] Alexander S Gillis and Valerie Silverthorne. What is integrated development environment (ide)?, Sep 2018.

[15] GitKraken. Gitlens, May 2023.

[16] GNU and Free Software Foundation. What is gdb?, May 2023.

[17] Kings Programming Education Tools Group, Sep 2022.

[18] Red Hat. Language support for java(tm), March 2023.

[19] David Intersimone. Antique software: Turbo pascal v1.0. https://web.archive.org/web/20101221211755/http://edn.embarcadero.com/article/20693, December 2010.

[20] JetBrains. The state of developer ecosystem in 2019 infographic, 2019.

[21] JetBrains. Download intellij idea, Mar 2023.

[22] Mike Jones, Gordon Hogenson, J. Martens, Tom Pratt, William Anton Rohm, Genevieve Warren, Kraig Brockschmidt, Beth Harvey, Oscar Sun, and Kaycee. First look at the visual studio debugger, Sep 2022.

[23] John G. Kemeny and Thomas E. Kurtz. Dartmouth time-sharing. *Science*, 162(3850):223–228, 1968.

[24] Thomas E. Kurtz. *BASIC*, page 515–537. Association for Computing Machinery, New York, NY, USA, 1978.

[25] Bell Laboratories. *Unix Programmer's Manual*, volume 2. Bell Telephone Laboratories Inc., 7 edition, 1983.

[26] Mark A. Linton. The evolution of dbx. In *USENIX Summer*, 1990.

[27] Norman S. Matloff and Peter Jay Salzman. *The art of debugging with GDB and DDD: For professionals and students*. No Starch Press, 2008.

[28] Microsoft. Microsoft announces visual studio 97, a comprehensive suite of microsoft visual development tools, Jan 1997.

[29] Microsoft. Debug adapter protocol, 2021.

[30] Microsoft. Language server protocol, 2022.

[31] Microsoft. Download visual studio code, Apr 2023.

[32] Microsoft. Extension api, May 2023.

[33] Microsoft. Extensions for visual studio code, 2023.

[34] Microsoft. Publishing extensions. https://code.visualstudio.com/api/working-with-extensions/publishing-extension, Jun 2023.

[35] Microsoft. Why visual studio code?, May 2023.

[36] Stack Overflow. Stack overflow developer survey 2022, May 2022.

[37] Robert Plant and Stephen Murrell. *An executive's Guide to Information Technology: Principles, business models, and terminology*. Cambridge University Press, 2007.

[38] Maya Posch. Revisiting borland turbo c and c++, Apr 2023.

[39] RevDeBug. Debugging history - origins, Feb 2020.

[40] Kateryna Shyniaieva. The most popular ides for developers in 2023, Feb 2023.

[41] Sarika Sinha. Debugging the eclipse ide for java developers, Jun 2017.

[42] TypeScript. What is typescript?, 2023.

[43] Slava Vaniukov. 16 best ides for software development: Overview for 2023, Apr 2023.

[44] Veracode. What is ide or integrated development environments?, Dec 2020.

[45] Tom Wadlow. Turbo pascal. *Byte*, 9(7):267–278, Jul 1984.

[46] Martin H. Weik. *compile-and-go*, pages 260–260. Springer US, Boston, MA, 2001.

[47] Yeoman. Getting started with yeoman. https://yeoman.io/learning/, 2023.

[48] Qin Zhao, Saman Amarasinghe, Rodric Rabbah, Larry Rudolph, and Weng Wong. How to do a million watchpoints: Efficient debugging using dynamic instrumentation. volume 4959, 03 2008.

Appendix A Snail Language Support package.json

The following is the package.json file from Snail Language Support.

```
{
  "name": "snail-language-support",
  "displayName": "Snail Language Support",
  "publisher": "cprein19",
  "description": "Extension providing useful language support for the Snail
      programming language",
  "version": "1.0.0",
  "engines": {
    "vscode": "^1.71.0"
  },
  "categories": [
    "Programming Languages",
    "Snippets"
  ],
  "activationEvents": [
    "onLanguage:snail",
    "onCommand:snail-language-support.runSnailFile"
  ],
  "main": "./client/out/extension",
  "contributes": {
    "languages": [
      {
        "id": "snail",
        "aliases": [
          "Snail",
          "snail"
        ],
        "extensions": [
          ".sl"
        ],
        "configuration": "./configurations/language-configuration.json",
        "icon": {
          "light": "./images/snail.png",
          "dark": "./images/snail.png"
        }
      }
    ],
    "grammars": [
      {
        "language": "snail",
        "scopeName": "source.sl",
        "path": "./syntaxes/snail.tmLanguage.json"
      }
    ],
    "snippets": [
      {
        "language": "snail",
        "path": "./snippets/snippets.json"
      }
    ],
    "commands": [
```

```json
      {
        "command": "snail-language-support.runSnailFile",
        "title": "Run Snail Program",
        "category": "Snail",
        "enablement": "!inDebugMode",
        "icon": "$(play)"
      }
    ],
    "menus": {
      "commandPalette": [
        {
          "command": "snail-language-support.runSnailFile",
          "when": "resourceLangId == snail"
        }
      ],
      "editor/title/run": [
        {
          "command": "snail-language-support.runSnailFile",
          "when": "resourceLangId == snail",
          "group": "navigation@1"
        }
      ]
    },
    "configuration": {
      "type": "object",
      "title": "Snail Language Support",
      "properties": {
        "snailLanguageServer.maxNumberOfProblems": {
          "scope": "resource",
          "type": "number",
          "default": 100,
          "description": "Controls the maximum number of problems produced by the server."
        },
        "snailLanguageServer.trace.server": {
          "scope": "window",
          "type": "string",
          "enum": [
            "off",
            "messages",
            "verbose"
          ],
          "default": "off",
          "description": "Traces the communication between VS Code and the language server."
        },
        "snailLanguageServer.snailPath": {
          "scope": "application",
          "type": "string",
          "default": "snail",
          "description": "Path to snail executable to use for Language Server"
        }
      }
    },
    "breakpoints": [
      {
        "language": "snail"
      }
    ]
  },

  "scripts": {
    "vscode:prepublish": "npm run compile",
    "compile": "tsc -b",
    "lint": "eslint ./client/src --ext .ts,.tsx",
    "postinstall": "cd client && npm install && cd ..",
    "test": "sh ./scripts/e2e.sh"
  },
  "devDependencies": {
    "@types/glob": "^8.0.1",
    "@types/mocha": "^9.1.0",
    "@types/node": "^16.11.7",
    "@typescript-eslint/eslint-plugin": "^5.30.0",
    "@typescript-eslint/parser": "^5.30.0",
    "eslint": "^8.13.0",
    "mocha": "^9.2.1",
    "typescript": "^4.8.4"
  },

  "repository": {
    "type": "git",
    "url": "https://github.com/snail-language/snail-language-support"
  },
  "bugs": {
    "url": "https://github.com/snail-language/snail-language-support/issues"
  }
}
```