JOHANNES KEPLER
UNIVERSITY LINZ

Author
Lukas Aichhorn

Submission
Institute for System Software

Thesis Supervisor
Dipl.-Ing. Dr. Markus Weninger, BSc

October 2023

Visual Studio Code Extension for Siemens G-Code

Bachelor’s Thesis
to confer the academic degree of
Bachelor of Science
in the Bachelor’s Program
Informatik

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Str. 69
4040 Linz, Austria
www.jku.at
DVR 0093696

JOHANNES KEPLER
UNIVERSITY LINZ

Dipl.-Ing. Dr. Markus Weninger, BSc
Institute for System Software
T +43-732-2468-4361
markus.weninger@jku.at

Bachelor's Thesis
Sinumerik VS Code Extension

Student: Lukas Aichhorn
Advisor: Dipl.-Ing. Dr. Markus Weninger, Bsc
Company Advisor: Gerald Kettl, Fa. Fill
Start date: March 2023

Mit Hilfe eines VS Code Plugin soll eine zeitgemäße Entwicklungsumgebung für Siemens Sinumerik Steuerungen geschaffen werden.
Alle systemrelevanten Funktionen, Operanden, Maschinendaten und Variablen sollen in diesem Plugin per IntelliSense angeboten werden und der Syntax farblich hervorgehoben werden.
Über Tooltips soll bei Verwendung der Funktionen, Maschinendaten und Variablen im Code eine entsprechende Hilfe angeboten werden.
Über „Dokument formatieren“ soll eine automatische Formatierung hinsichtlich Zeilenumbrüche und Einzüge bei Verschachtelungen und Leerzeichen vor und nach Operanden erfolgen. Außerdem soll eine Prüfung auf den Fill NcStyleGuide erfolgen.
Beim VSCode-Befehl „Ausführen und Debuggen“ soll die gesamte Syntax des Workspace geprüft werden. Anschließend findet eine Synchronisierung zwischen Workspace am Programmiergerät und dem Workspace in der Steuerung statt.

Auflistung der einzelnen Aufgabenpunke:

- Erhebung der Daten aus der offiziellen Dokumentation und Siemensdateien (Funktionen, Operanden, Maschinendaten, Variablen,…)
- Erstellung eines Parser zum einlesen der Daten
- Grammatik für die Sprache erstellen
- Syntax Highlighting
- Tooltips
- Dokument formattieren
- Workspace auf Grammatik prüfen (“Syntaxfehlererkennung”)
- Synchronisation per SSH mit der Steuerung
- Obfuskierung der Extension

<u>Modalities:</u>
The progress of the project should be discussed at least every four weeks with the advisor. A time schedule and a milestone plan must be set up within the first 3 weeks and discussed with the advisor. It should be continuously refined and monitored to make sure that the thesis will be completed in time. The final version of the thesis must be submitted not later than 31.08.2023.

Abstract

Visual Studio Code (VS Code) can be a powerful and versatile code editor. However, Microsoft achieves this by using so-called extensions. Extensions are small modular programs with which you can easily customise your editor. Nevertheless, this system can have a big flaw – the lack of an extension for your particular needs. Since this becomes apparent when working with *G-Code* while using VS Code, this thesis will attempt to create an extension that remedies this issue and supports any developers working with this language in as many ways as possible.

Kurzfassung

Visual Studio Code (VS Code) kann ein extrem mächtiger und vielseitiger Code Editor sein. Dies ist jedoch nur realisiert durch sogenannte Extensions. Extensions sind kleine modulare Programm, mit deren Hilfe man seinen Editor exakt nach seiner Vorstellung gestalten kann. Trotzdem kann dieses System große Fehler aufweisen – durch das Fehlen von Extensions, welche die geforderten Bedürfnisse abdecken. Da dies beim Arbeiten mit *G-Code* auffällig wird, versucht diese Arbeit eine Extension zu erstellen, welche dieses Problem behebt und alle Entwickler welche mit dieser Sprache arbeiten unterstützt.

Table of Content

Contents

1 Introduction

VS Code can be a potent tool for every developer since it offers many advantages over other editors. The most important one is its modularity. With the help of plugins, this enables VS Code to be the most versatile code editor, which allows developers to customise VS Code. Yet customisation is not the only advantage this brings. Furthermore, plugins enable VS Code to work with any Language that has a plugin developed - in contrast to most viable competitors, which only specialise their editor for a single language.

However, despite VS Code offering tens of thousands of plugins, every plugin offered for G-Code is very lacklustre, with either limited features or very buggy implementations. Because of this, using VS Code while developing G-Code can be very frustrating and inefficient since a good code editor can massively boost productivity and may even reduce the occurrence of errors in your code. Therefore, this thesis will focus on creating a new plugin for Siemens G-Code to solve the problem of inadequate plugins.
Our contributions encompass:

- a web scraper which generates data from the official Siemens documentation sources
- custom providers which generate workflow-supporting features
- a Regex-based approach to tokenising and syntax highlighting
- the attempt at creating a grammar from which you can generate a Scanner and a Parser to gain an improved syntax highlighting experience and further introduce a syntax error reporting system

2 Background

2.1 G-Code

G-Code is a language mostly for computer-aided manufacturing, e.g. in 3D printing, but more prevalent in machines to instruct a cutting tool where to move and when to cut. Furthermore, G-Code has many implementations, and one of the most commonly used is Siemens Sinumerik. [7]

Because of the age of this language, G-Code still uses GO-TO statements to write reusable code. This concept works by creating a named label; whenever you want to execute this code, you "jump" to the position the label is at, and the machine runs the code. Despite this usefulness, one should use GO-TO statements with care since they can lead to horrible code if a programmer abuses them.

2.2 Obfuscation

Obfuscation is a potent tool for any developer or company trying to protect their source code. Obfuscation is a concept that takes your existing readable source code and makes it unreadable to the human eye. Most of the time, this is achieved by renaming variables to something random, refactoring the code to a singular line and sometimes introducing random method calls that do not affect the code but complicate the execution path for anyone trying to decipher your code.

2.3 VS Code and the VS Code API

VS Code is a prominent source code editor developed with the electron framework. This is important because this means we must develop the plugin in JavaScript or TypeScript. As a result, the VS Code API is also written in the same languages. [2] Because of personal preferences, the plugin was developed in Typescript.

The VS Code API is extensive and contains hundreds, if not thousands, of different methods for any plugin. While this may seem intimidating, the documentation by VSCode makes working with the API very intuitive. [5] Also, the most important concepts for this thesis are events, command registering and providers, which are the only ones we will discuss in detail.

2.3.1 Command Registering

Command registering is essential for any feature that has to be activated by the user and should not work automatically, e.g. activating a feature via a shortcut key. Command registering works by first declaring the name of your command into the package.json file. Afterwards, the plugin's source code must register the command with the VS Code API while the extension is activated. Consequently, the editor recognises the command, enabling the user to execute it willingly.

2.3.2 Events

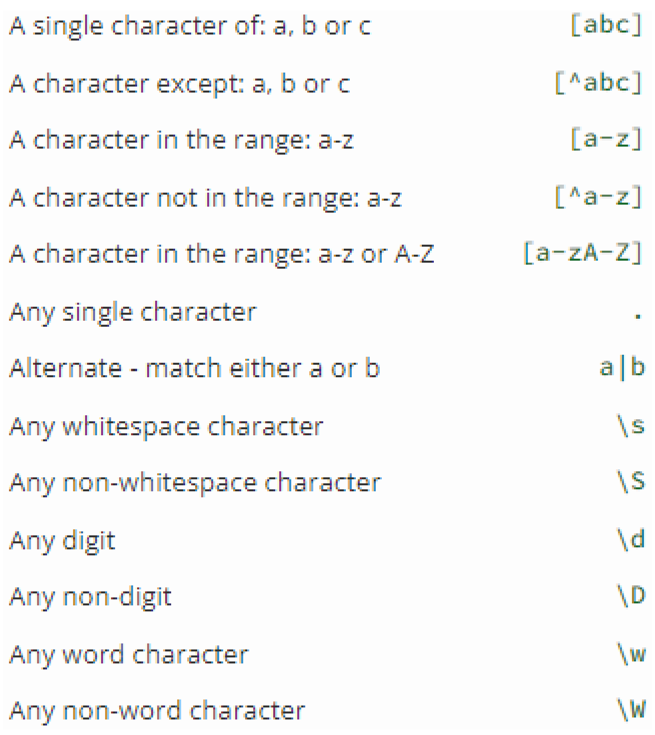
Events are also crucial to working with the VS Code API since this enables developers to react to user input. The names of the events follow the pattern "on[Will|Did]VerbNoun?" and cover any possible user input to trigger the event.

2.3.3 Providers

We use Providers for communicating back to the VS Code API to get the desired results. Once a provider registers with the API, it may be triggered automatically or manually via an event, depending on the type of provider that is needed or used. Once a provider is triggered, it will call a designated function assigned while registering. After calling the method, most Providers will return a value. Next, this value is passed to the API, which then handles further processing for you.

2.4 Regex

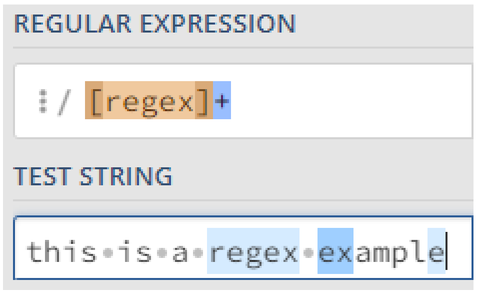
Regular expressions, more commonly known as Regex, are an important concept in computer science. Regex are a set of rules which let you specify patterns to find matching patterns in a string. For our purposes, we will only need basic regex concepts.



A single character of: a, b or c	[abc]
A character except: a, b or c	[^abc]
A character in the range: a-z	[a-z]
A character not in the range: a-z	[^a-z]
A character in the range: a-z or A-Z	[a-zA-Z]
Any single character	.
Alternate - match either a or b	a|b
Any whitespace character	\s
Any non-whitespace character	\S
Any digit	\d
Any non-digit	\D
Any word character	\w
Any non-word character	\W

Figure 1: The most important basic regex rules.[3]

It is possible to match nearly every text pattern using the rules from Figure 1.



```
/ [regex]+
```

```
this is a regex example
```

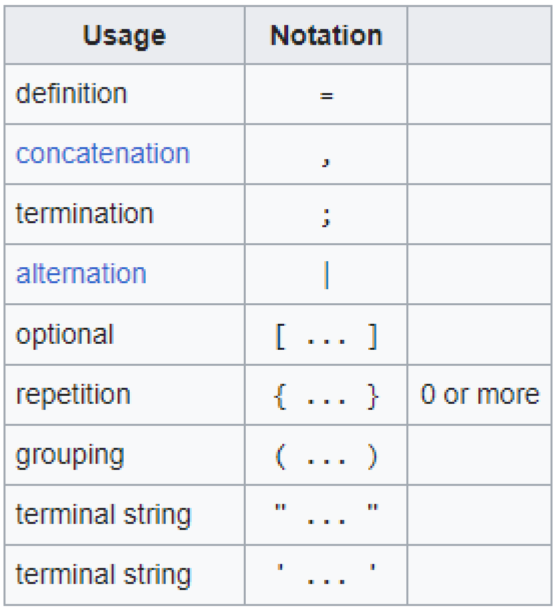
Figure 2: a simple regex example with the matches in the test string highlighted in blue. [3]

Figure 2 shows a small example of using a regex. The square brackets indicate a set, and the plus sign indicates one or more matches in a matching group. Furthermore, using these concepts, one can create regexes that match entire sentences or are as precise as required.

2.5 Compiler-construction basics

2.5.1 EBNF

The *Extended Backus Naur Form*, more commonly known as EBNF, is a set of rules to specify a grammar.



Usage	Notation	
definition	=	
concatenation	,	
termination	;	
alternation	|	
optional	[...]	
repetition	{ ... }	0 or more
grouping	(...)	
terminal string	" ... "	
terminal string	' ... '	

Figure 3: The EBNF rules as proposed in an ISO standard. [6]

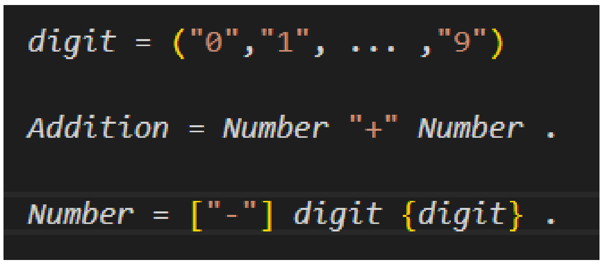
Conceptually, EBNF and Regex are not too far apart; computer scientists can use both to "split up" text in a certain way and extract information.

2.5.2 Tokens

However, EBNF does not try to produce matches but tokens. Tokens are small elementary strings that have a unique assigned meaning. Programs then further process these tokens to map input to certain predefined token classes.

2.5.3 Grammars

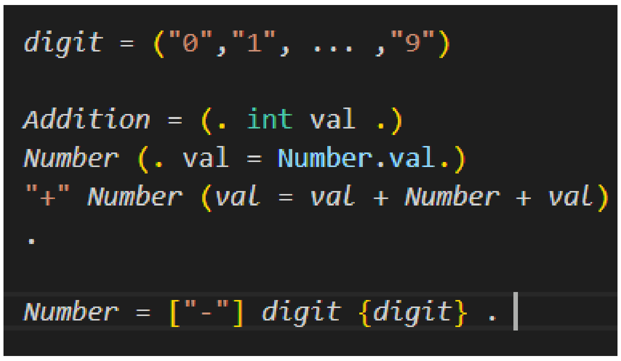
However, a program must have a predefined grammar to use the full extent of tokens. A Grammar is an instruction on how to form valid sentences in a (in our case) programming language. To accomplish this a, grammar consists of multiple productions. A single production can consist of either *Terminal Symbols* i.e. other productions or *Non Terminal Symbols* i.e. atomic Tokens.



```
digit = ("0","1", ... ,"9")

Addition = Number "+" Number .

Number = ["-"] digit {digit} .
```

Figure 4: A small example for a valid example grammar.

In addition, there are also *Attributed Grammars* or ATGs, which are used to improve grammars to do further work with results from a given grammar. This works by taking input values or returning output values for special tokens.



```
digit = ("0","1", ... ,"9")

Addition = (. int val .)
Number (. val = Number.val.)
"+" Number (val = val + Number + val)
.

Number = ["-"] digit {digit} .
```

Figure 5: The same as Figure 4 but attributed.

2.5.4 Coco

Coco is a compiler generator which describes a tool that takes an ATG as input and generates a scanner and a recursive descent parser. This is helpful since after completing the ATG, which would also be necessary for the manual approach, you can easily use it to do all the work.

3 Web Scraper

This section discusses the web scraper we used in conjunction with the extension. Therefore, it is not part of the extension itself but provides processed data for the extension to process. Thus, the scraper is only executed when there is a change in the input sources to apply these differences.

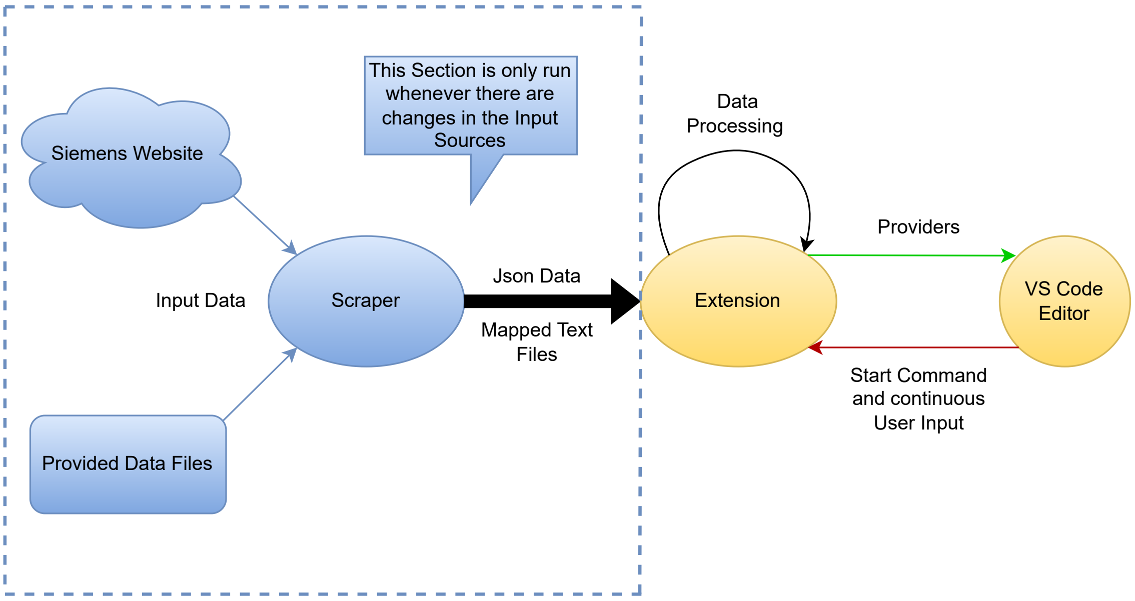
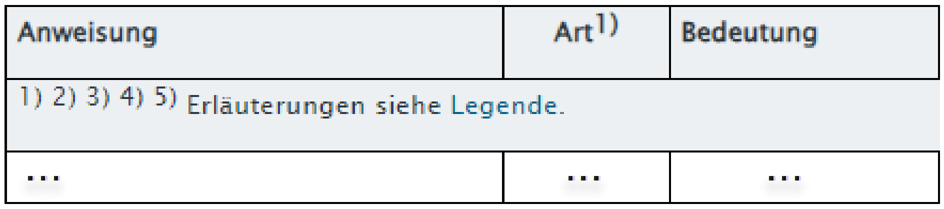


Figure 6: A rough overview over the entire system of the extension; the section inside the blue square represents the entire scraper system.

3.1 Siemens Website

The website contains every single command for the Siemens G-Code Language in a single table. The idea is to get the Website's HTML Code, filter the result for the usable sections, and then store them accordingly.



Anweisung	Art 1)	Bedeutung
1) 2) 3) 4) 5) Erläuterungen siehe Legende.		
...	...	...

Figure 7: A small website excerpt with the most important categories. [4]

We realised this by using selenium to open a browser and then a JavaScriptExector since the table data is loaded dynamically via Javascript and, therefore, unavailable immediately.

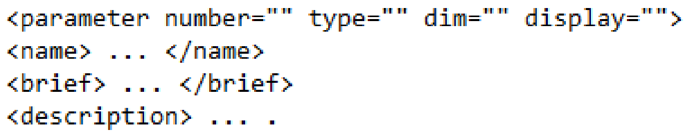
Next, we navigate the table data using XPATH and depending on the contents, the nodes are saved to different files for later use. The different types are:

- Operators - a list of all commands considered operators by Siemens that are useful for the auto-formatting feature
- Descriptions - a file that maps the name of the command to the description given by Siemens and later provides useful hover information
- TMGrammar - maps every command of the language to a specific group in JSON-Format together with a basic predefined Regex Structure, which is beneficial for generating the tokens for the Regex version of Syntax Highlighting

Further detail on each file and how we use them is provided in Section 5 and Section 6.1.

3.2 Documentation Files

Since the website is not exhaustive, we use certain Documentation files to supplement the input data. There are multiple types of files however, for simplicity and brevity, we are only going to look at one specific type that is the most general.
The .mdat files (short for machine data) have a specified format being:



```
<parameter number="" type="" dim="" display="">
<name> ... </name>
<brief> ... </brief>
<description> ... .
```

Figure 8: Examplary format of a .mdat file.

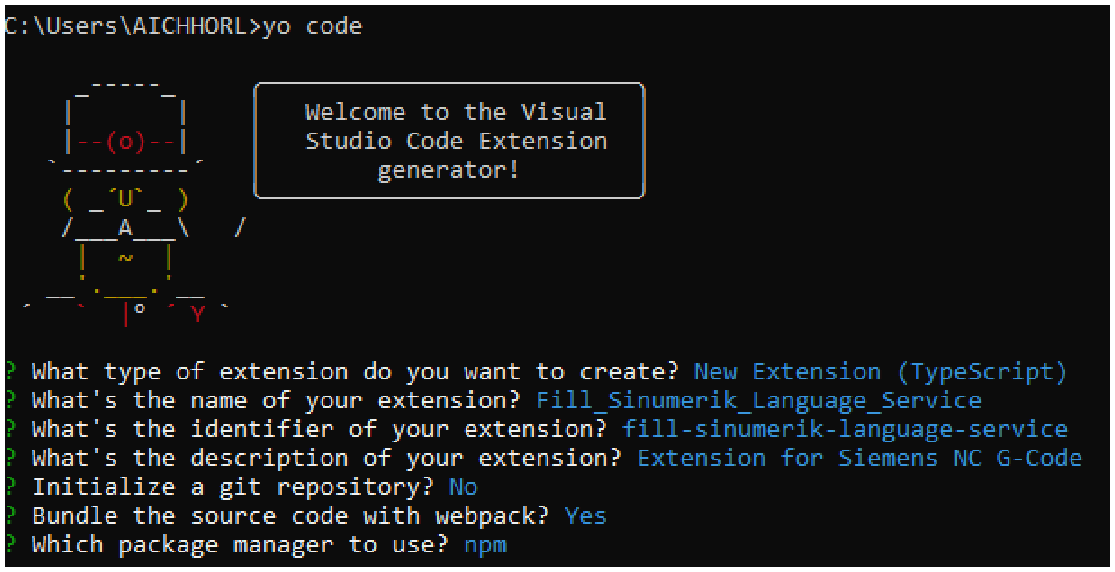
As a result of this strict formatting, it is relatively easy to write a program that takes all these files as input and consequently generates more data that is then once again added to the three files mentioned in section Section 3.1

4 Extension Initialisation and Packaging

This section focuses on the fundamental structures of VS Code Extension Development, i.e. how to start developing your extension using and creating your finished extension after you finish your development process using vsce.

4.1 yeoman and creating a VS Code Extension

Yeoman is a generator tool that allows developers to quickly and easily generate different code scaffolds, which enables users to set up projects without having to do much of the often tedious and complex setup when using multiple tools. It achieves this by applying modular principles to its generators. Since this means that it should be effortless to create a generator, many languages, tools and plugins have decided to do so. Hence, VS Code has also created a generator for its needs. It mainly consists of the build configuration, packaging service and the package manager [8].



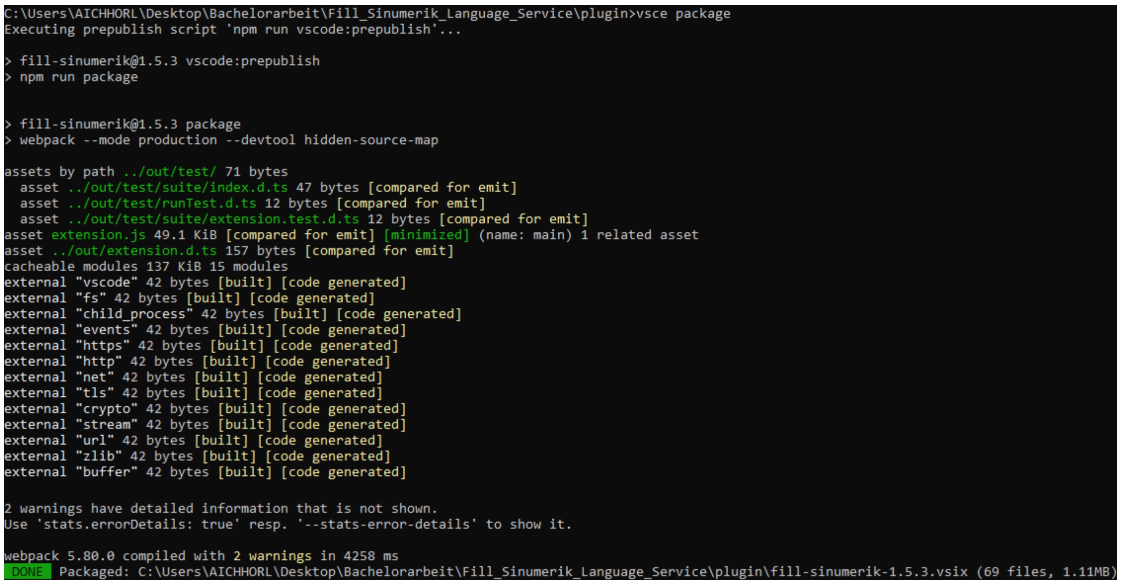
```
C:\Users\AICHHORL>yo code

     _-----_     ╭──────────────────────────╮
    |       |    │   Welcome to the Visual  │
    |--(o)--|    │   Studio Code Extension  │
   `---------´   │        generator!        │
    ( _´U`_ )    ╰──────────────────────────╯
    /___A___\   /
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

? What type of extension do you want to create? New Extension (TypeScript)
? What's the name of your extension? Fill_Sinumerik_Language_Service
? What's the identifier of your extension? fill-sinumerik-language-service
? What's the description of your extension? Extension for Siemens NC G-Code
? Initialize a git repository? No
? Bundle the source code with webpack? Yes
? Which package manager to use? npm
```

Figure 9: An example of how to generate a new extension using the Yo Code Generator.

For our purposes, creating a new extension will suffice however, there are two main choices: Whether to use npm or yarn and whether to use webpack or not. While the npm versus yarn debate is mostly a matter of taste and does not change the extension's outcome significantly, using webpack brings the advantage of automatically obfuscating your code (i.e., making your source code unreadable to the human eye). Obfuscation can be imperative because there are easy methods to get the source code of any VS Code extension within a few minutes. The disadvantage is that sometimes webpack can be more confusing, but this should not be an issue unless you plan to stray from the norm when trying to package.

4.2 Extension packaging and vsce

After the development process finishes and your code works, it is time to package your extension into a format that VS Code can handle to install the extension on any machine. VS Code recommends using the *Visual Studio Code Extension Manager (vsce)*. The manager runs and activates scripts already included in the yeoman scaffolding and builds the code into a usable form for any VS Code user.



```
C:\Users\AICHHORL\Desktop\Bachelorarbeit\Fill_Sinumerik_Language_Service\plugin>vsce package
Executing prepublish script 'npm run vscode:prepublish'...

> fill-sinumerik@1.5.3 vscode:prepublish
> npm run package


> fill-sinumerik@1.5.3 package
> webpack --mode production --devtool hidden-source-map

assets by path ../out/test/ 71 bytes
  asset ../out/test/suite/index.d.ts 47 bytes [compared for emit]
  asset ../out/test/runTest.d.ts 12 bytes [compared for emit]
  asset ../out/test/suite/extension.test.d.ts 12 bytes [compared for emit]
asset extension.js 49.1 KiB [compared for emit] [minimized] (name: main) 1 related asset
asset ../out/extension.d.ts 157 bytes [compared for emit]
cacheable modules 137 KiB 15 modules
external "vscode" 42 bytes [built] [code generated]
external "fs" 42 bytes [built] [code generated]
external "child_process" 42 bytes [built] [code generated]
external "events" 42 bytes [built] [code generated]
external "https" 42 bytes [built] [code generated]
external "http" 42 bytes [built] [code generated]
external "net" 42 bytes [built] [code generated]
external "tls" 42 bytes [built] [code generated]
external "crypto" 42 bytes [built] [code generated]
external "stream" 42 bytes [built] [code generated]
external "url" 42 bytes [built] [code generated]
external "zlib" 42 bytes [built] [code generated]
external "buffer" 42 bytes [built] [code generated]

2 warnings have detailed information that is not shown.
Use 'stats.errorDetails: true' resp. '--stats-error-details' to show it.

webpack 5.80.0 compiled with 2 warnings in 4258 ms
 DONE  Packaged: C:\Users\AICHHORL\Desktop\Bachelorarbeit\Fill_Sinumerik_Language_Service\plugin\fill-sinumerik-1.5.3.vsix (69 files, 1.11MB)
```

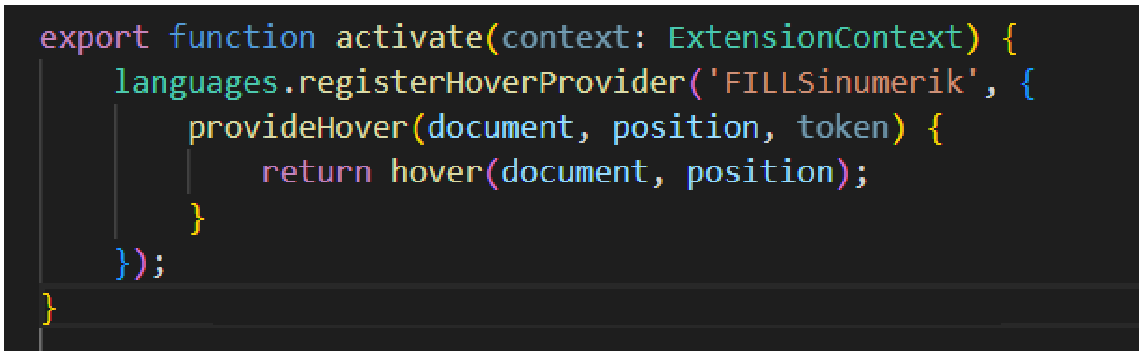
Figure 10: An exemplary packaging process using vsce via webpack.

5 Providers

Other than Syntax Highlighting, Providers are the centrepiece of any VS Code extension functionality. The VS Code API uses this concept to let you program in an event-based way all on its own since, depending on the provider you are implementing, your function is called on its own after registering itself with the API. This lets you focus on the functionality and makes working with the VS Code API easy and intuitive.

5.1 General Providers

The general concept of providers is as follows. Each VS Code extension has an extension.ts (or .js if using JavaScript instead of TypeScript) file, which is the interface between VS Code and your Extension. Every extension.ts file has an activate function that runs once the extension is activated. It is here that every provider has to be registered.



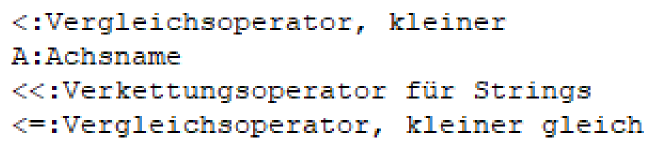
```
export function activate(context: ExtensionContext) {
    languages.registerHoverProvider('FILLSinumerik', {
        provideHover(document, position, token) {
            return hover(document, position);
        }
    });
}
```

Figure 11: An example of how to register a provider.

As seen in Figure 11, registration is accomplished via the languages namespace of the VS Code API. First, in this example, you call the registerHoverProvider method, which expects the inputs of 'DocumentSelector' and 'Provider'. In this case, DocumentSelector means the ID of the language to use, while a Provider could be as simple as returning a static string, it can also be a function that determines the result the provider receives. Finally, the provider is called after being successfully registered whenever the VS Code editor hovers over something. One can apply this to all providers with differences in providing methods.

5.2 Hover Provider

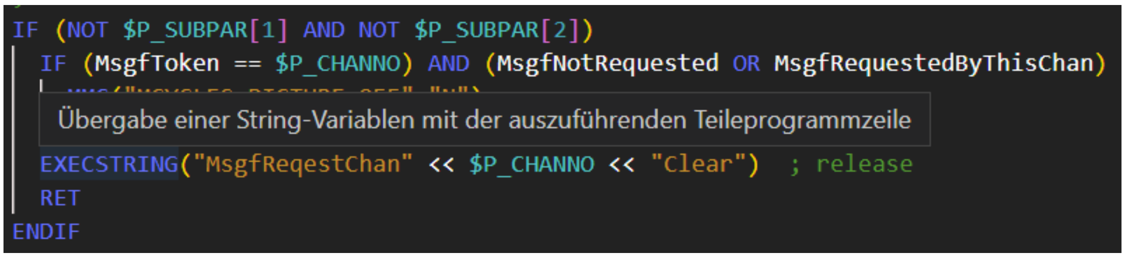
The hover provider is a provider that is called when your mouse cursor hovers over a piece of code. In our case, this is exceptionally useful because of the overwhelming number of instructions in the G-Code Language and thus makes it nearly impossible to remember what every one of them does. To remedy this, we display the description from the instruction documentation while hovering over them. As a point of reference for the information that needs to be displayed, we have already prepared the Descriptions file in Section 3.1.



```
<:Vergleichsoperator, kleiner
A:Achsname
<<:Verkettungsoperator für Strings
<=:Vergleichsoperator, kleiner gleich
```

Figure 12: An example for the structure of the Descriptions file.

Therefore, we only have to read the file content to get our matching description for the hovered content. Despite this, since we do not receive information about which word is highlighted but only information about the cursor's position, we still have to find out which word we hover over. We achieve this using the TextDocument.getWordRangeAtPosition() method from the VS Code API, which takes our given position and returns the sought-after string. Finally, we map the string to a word and get our description, which we return to the Hover Provider.

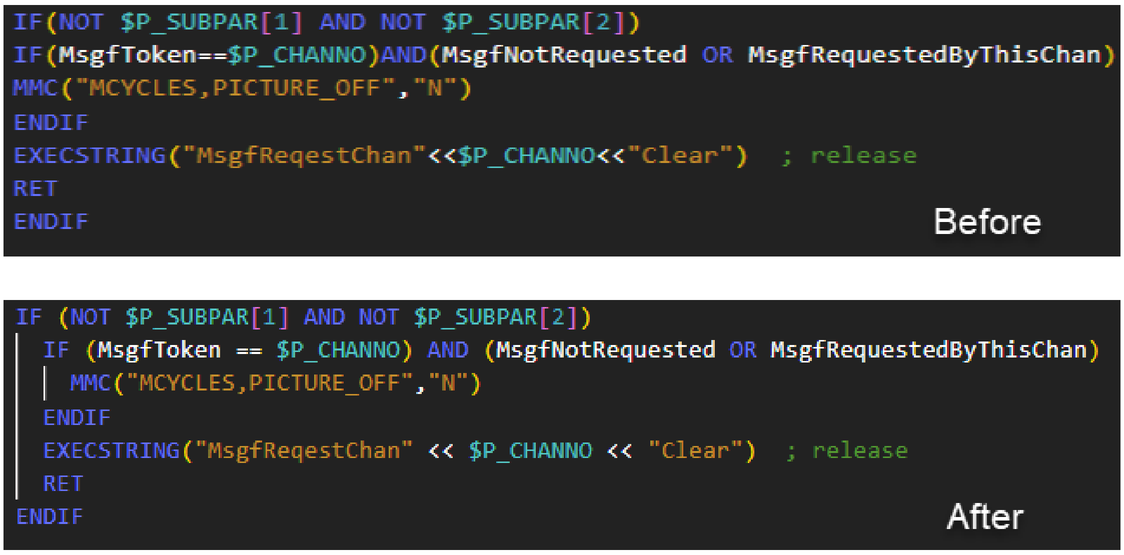


```
IF (NOT $P_SUBPAR[1] AND NOT $P_SUBPAR[2])
  IF (MsgfToken == $P_CHANNO) AND (MsgfNotRequested OR MsgfRequestedByThisChan)
   Übergabe einer String-Variablen mit der auszuführenden Teileprogrammzeile
  EXECSTRING("MsgfReqestChan" << $P_CHANNO << "Clear")  ; release
  RET
ENDIF
```

Figure 13: The result, while hovering over EXECSTRING, displays a tooltip generated from the Siemens Website data.

5.3 DocumentFormattingEdit Provider

Sometimes, programmers can be a bit lazy. To counteract this, we implemented a feature that catches lazy programming, e.g. "1+ 1 ;", "A = 3;". Consequently, we implemented a DocumentFormattingEdit Provider, which expects the entire TextDocument as the input. The TextDocument is now being read line by line, and the plugin checks for either keywords such as "for" or "while" or any operators specified in the Operators file created in Section 3.1. Should any matches be found, the program checks if indentations and whitespaces are all correct - if any "mistakes" are encountered, the provider creates a DocumentEdit, which contains the position of the "mistake" to be replaced and the string with which to replace it. Finally, the VS Code API handles the rest.



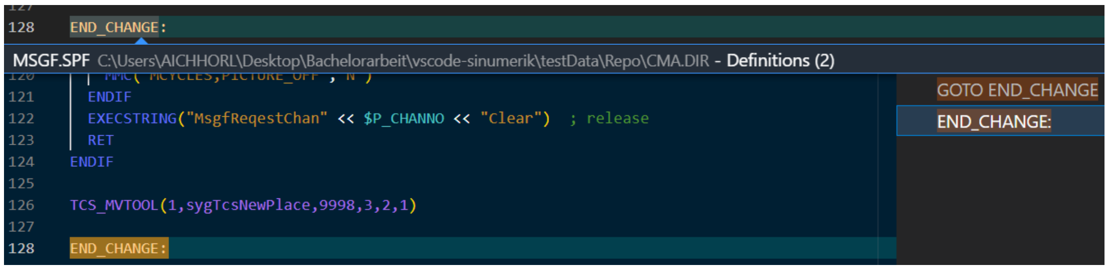
```
IF(NOT $P_SUBPAR[1] AND NOT $P_SUBPAR[2])
IF(MsgfToken==$P_CHANNO)AND(MsgfNotRequested OR MsgfRequestedByThisChan)
MMC("MCYCLES,PICTURE_OFF","N")
ENDIF
EXECSTRING("MsgfReqestChan"<<$P_CHANNO<<"Clear")  ; release
RET
ENDIF
```

```
IF (NOT $P_SUBPAR[1] AND NOT $P_SUBPAR[2])
  IF (MsgfToken == $P_CHANNO) AND (MsgfNotRequested OR MsgfRequestedByThisChan)
    MMC("MCYCLES,PICTURE_OFF","N")
  ENDIF
  EXECSTRING("MsgfReqestChan" << $P_CHANNO << "Clear")  ; release
  RET
ENDIF
```

Figure 14: A program before and after formatting via the extension.

5.4 Definition Provider

Since G-Code works with GOTO statements, the ability to jump to the label is advantageous in reducing mindless searching. For this reason, we decided to implement a Definition Provider, which enables the user to see all possible relevant labels and jumps.
This works by once again getting the word at the cursor's position in the same way as with the Hover Provider in Section 5.2. In contrast to the Hover Provider, we now, however, check the entire document whether this line of the document contains this particular word similar to the DocumentFormattingEdit Provider in Section 5.3 and if there is a match we now have to push the *Uniform Resource Identifier (URI)* into an array together with its line-number. Next, this is resolved as a Promise and handed back to the VS Code API, which then creates the final menu for the feature.



```
END_CHANGE:
```

MSGF.SPF C:\Users\AICHHORL\Desktop\Bachelorarbeit\vscode-sinumerik\testData\Repo\CMA.DIR - Definitions (2)

```
  ENDIF
  EXECSTRING("MsgfReqestChan" << $P_CHANNO << "Clear")  ; release
  RET
ENDIF

TCS_MVTOOL(1,sygTcsNewPlace,9998,3,2,1)

END_CHANGE:
```

Figure 15: After pressing CTRL + Left Click, a window opens to display all possible places to go to

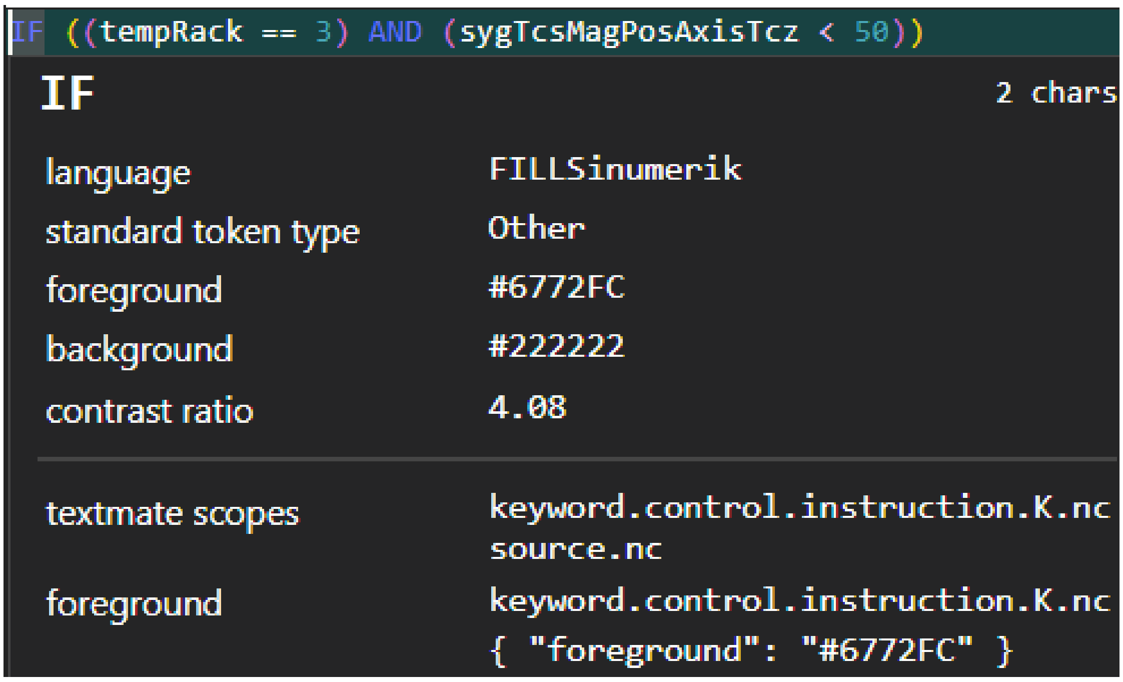
6 Tokenising

Tokenising goes hand in hand with Syntax Highlighting because to know how to highlight certain parts of code, the raw input has to be tokenised first, and according to these tokens, highlighting can be achieved by mapping the tokens to colours or possibly even different fonts et cetera.

6.1 Regex Approach

6.1.1 Tokenising using Textmate Grammars with Regex

The more traditional approach for Syntax Highlighting in VS Code Extensions is using Textmate Grammars. Despite its name, Textmate is not a real grammar in the traditional sense. However, it is a collection of Regexes in a JSON file fed to the Textmate Grammar Engine, which applies these rules to the document and returns a Tokenisation for the input.
We already supplemented a TextMate Grammar file called TMGrammar in Section 3.1 to implement this.

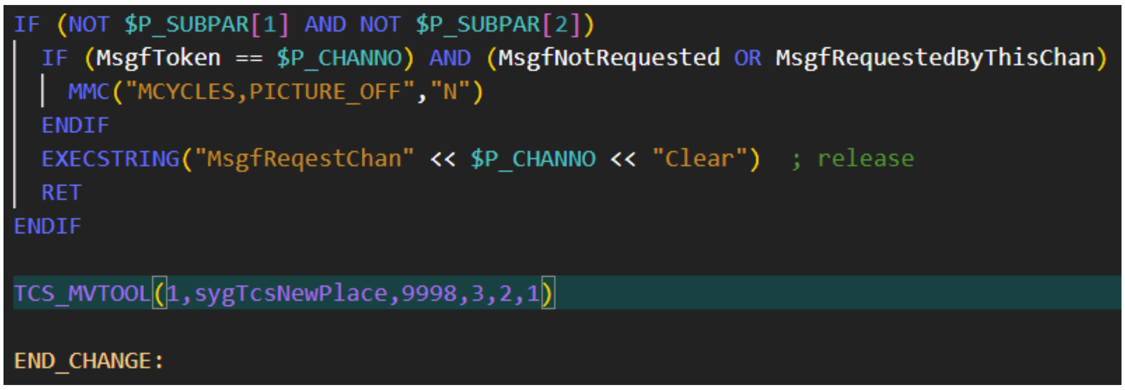


```
IF ((tempRack == 3) AND (sygTcsMagPosAxisTcz < 50))
```

IF	2 chars
language	FILLSinumerik
standard token type	Other
foreground	#6772FC
background	#222222
contrast ratio	4.08
textmate scopes	keyword.control.instruction.K.nc source.nc
foreground	keyword.control.instruction.K.nc { "foreground": "#6772FC" }

Figure 16: The assigned tokens for a piece of code illustrated via the VS Code token inspector.

As seen in Figure 16, this is now to tokenise the document and to group it into different groupings. These groupings are defined via the input of the "Art" (English → type) on the Siemens Website.



```
IF (NOT $P_SUBPAR[1] AND NOT $P_SUBPAR[2])
  IF (MsgfToken == $P_CHANNO) AND (MsgfNotRequested OR MsgfRequestedByThisChan)
    MMC("MCYCLES,PICTURE_OFF","N")
  ENDIF
  EXECSTRING("MsgfReqestChan" << $P_CHANNO << "Clear")  ; release
  RET
ENDIF

TCS_MVTOOL(1,sygTcsNewPlace,9998,3,2,1)

END_CHANGE:
```

Figure 17: Program code highlighted in the editor via the Regex Method.

6.1.2 Problems with the Regex Approach

Despite the ease of access and ease of use that Regex offers, when the Textmate Grammar gets too big, small changes to fix edge cases become almost impossible, even for the most experienced Regex users.

6.2 Grammar Approach

As a result, there was an attempt at creating a traditional grammar-based approach to remedy this problem. Despite investing a big chunk of time while working on this thesis, the scope of this problem could have been smaller to complete this within a reasonable timeframe. Therefore, we will be presenting our findings, which we have made up until the point at which we put this approach on hold for the moment and our ideas for what would have been the plan for the finalised version.

6.2.1 G-Code Grammar

Since there is no official Grammar for G-Code, we took the approach of reverse engineering the language into a Grammar. Together with a few people who have been working with G-Code for years. As a result, we created a Grammar with approximately 400 productions and 5000 tokens, which was going to be the basis for the rest of the Tokenisation.
The next step was generating a Scanner and Parser via Coco. However, this proved to be a challenge because of the huge size of the grammar and the limitations of most programming languages. This occurred because the maximum method size of most languages is around 65kB. The generated Parser had parts of code causing bytecode far bigger than 65kB; therefore, the machine could not run the program.
We remedied this by writing a script to split those methods in half or even thirds by altering the source code files. Because of further complications with the TypeScript version of Coco, we decided to change to the more familiar Java version, which then resolved persisting issues. However, this meant that now there would have to be a subprocess and to increase performance by not having to create a new Scanner and Parser every time the input is tokenised, we decided on a Webserver solution which would host the Scanner and Parser and communicate with the

extension via websockets. We came to this decision because of performance reasons since it would be wasteful to instantiate a new Scanner and Parser every time the user gave input, and instead, we reused the Scanner and Parser.

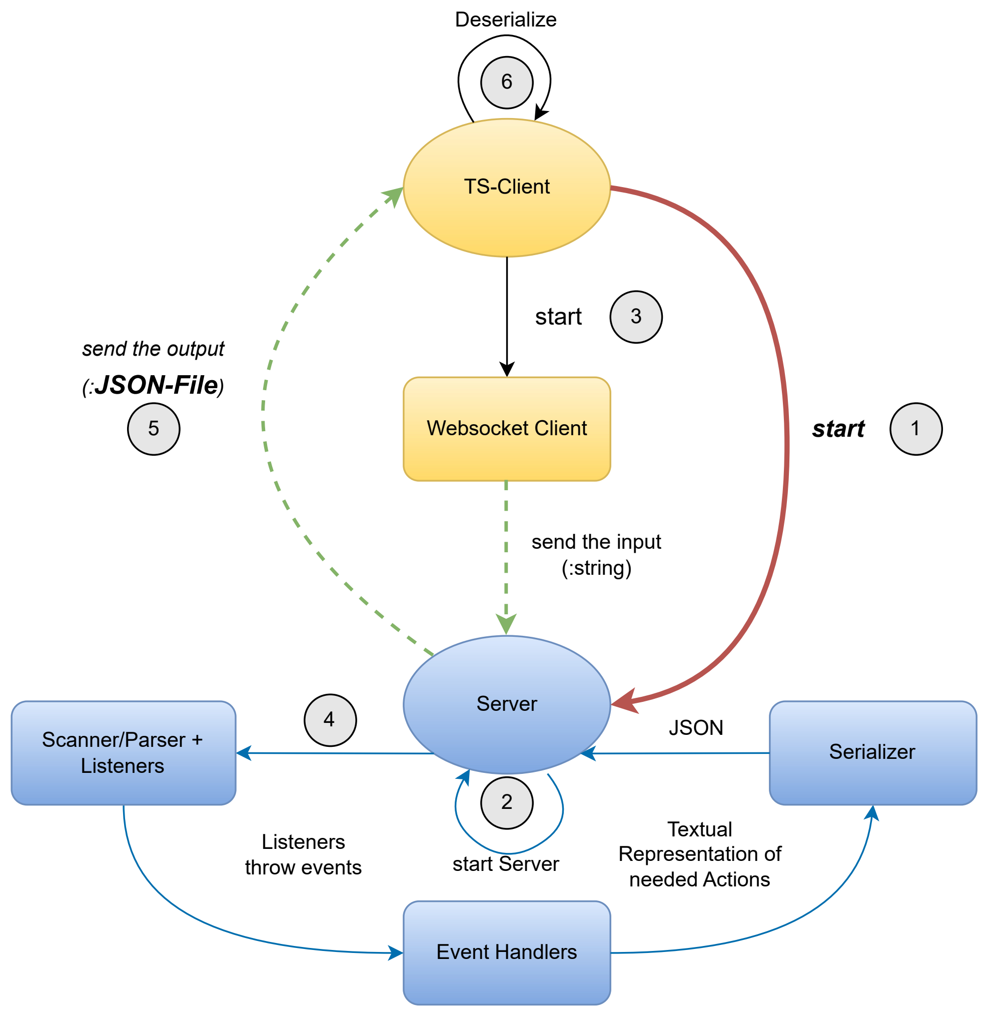
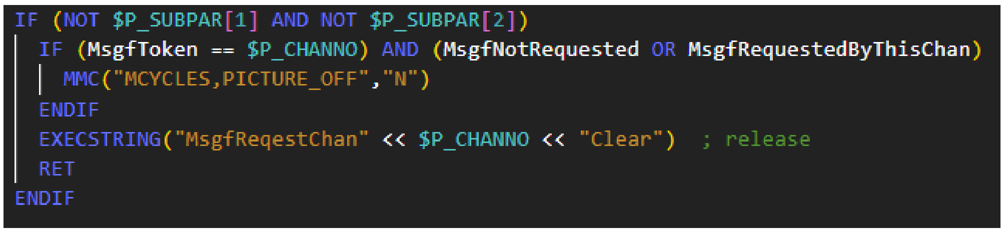


Figure 18: An overview of the communication between extension and server.

Figure 18 shows the general sequence of actions of the communication. First, the TS-Client (extension) signals the server to start itself. After the server has finished starting, the TS-Client starts a websocket Client, which connects to the server. Next, input is sent from the Client to the server and forwarded to the Scanner and Parser. Since the Scanner and Parser Grammar was attributed to include Listeneners for every Token, an event is thrown every time the scanner scans a token. As a result, the Event Handlers activate and, together with a JSON Serializer, transform the data into the needed JSON format. Finally, the server sends a JSON Message to the TS-Client, deserialising the message and processing the data.

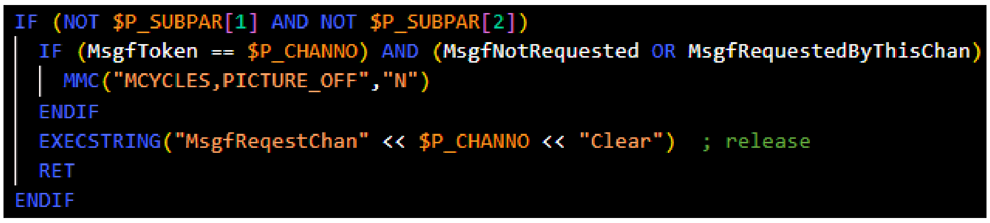
6.3 Theming

A further small part of this project was also theming. Since G-Code tokens differ from other languages, we created custom token groupings. For this reason, it was impossible to use popular colour themes, so we had to implement our own.



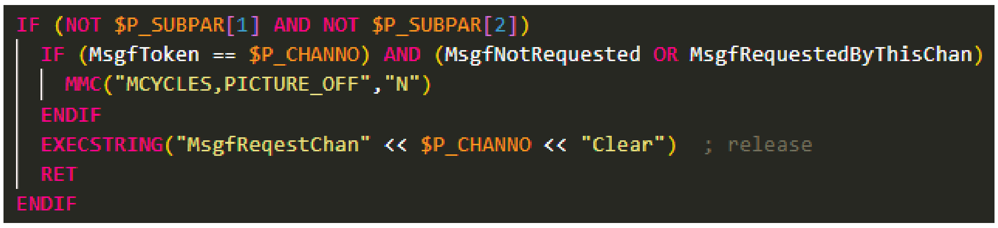
```
IF (NOT $P_SUBPAR[1] AND NOT $P_SUBPAR[2])
  IF (MsgfToken == $P_CHANNO) AND (MsgfNotRequested OR MsgfRequestedByThisChan)
    MMC("MCYCLES,PICTURE_OFF","N")
  ENDIF
  EXECSTRING("MsgfReqestChan" << $P_CHANNO << "Clear")  ; release
  RET
ENDIF
```

Figure 19: Sample code highlighted using the dark-style theme.



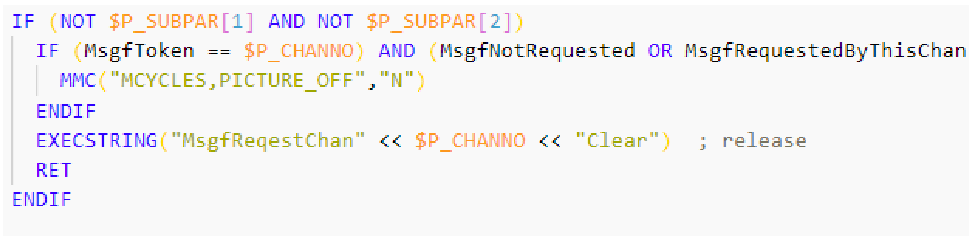
```
IF (NOT $P_SUBPAR[1] AND NOT $P_SUBPAR[2])
  IF (MsgfToken == $P_CHANNO) AND (MsgfNotRequested OR MsgfRequestedByThisChan)
    MMC("MCYCLES,PICTURE_OFF","N")
  ENDIF
  EXECSTRING("MsgfReqestChan" << $P_CHANNO << "Clear")  ; release
  RET
ENDIF
```

Figure 20: Sample code highlighted using the dark+-style theme.



```
IF (NOT $P_SUBPAR[1] AND NOT $P_SUBPAR[2])
  IF (MsgfToken == $P_CHANNO) AND (MsgfNotRequested OR MsgfRequestedByThisChan)
    MMC("MCYCLES,PICTURE_OFF","N")
  ENDIF
  EXECSTRING("MsgfReqestChan" << $P_CHANNO << "Clear")  ; release
  RET
ENDIF
```

Figure 21: Sample code highlighted using the monokai-style theme.



```
IF (NOT $P_SUBPAR[1] AND NOT $P_SUBPAR[2])
  IF (MsgfToken == $P_CHANNO) AND (MsgfNotRequested OR MsgfRequestedByThisChan
    MMC("MCYCLES,PICTURE_OFF","N")
  ENDIF
  EXECSTRING("MsgfReqestChan" << $P_CHANNO << "Clear")  ; release
  RET
ENDIF
```

Figure 22: Sample code highlighted using the light-style theme.

7 Overall Architecture

After discussing the components of the system, we are now looking at the architecture of the entire project.

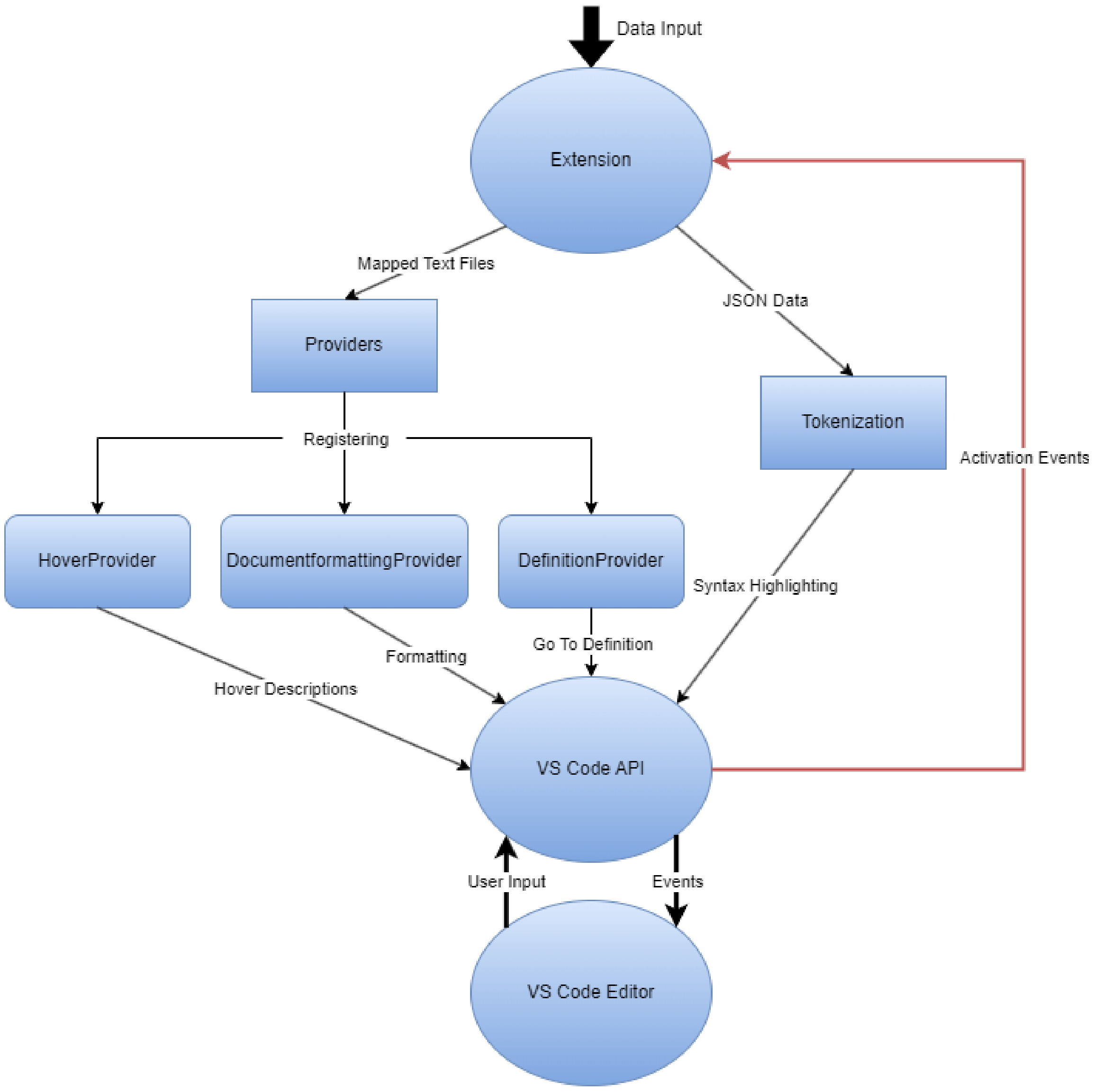


Figure 23: A detailed figure about the inner workings of the extension itself.

As evident from the chapter structure, the providers and the tokenisers are the two main components. However, while they work together to provide the plugin's functionalities, they work independently. The main interface is the VS Code API, which handles requests from the extension and sends activation events to the program to activate the functionalities and pass on the User Input from the editor.

8 Usage

Using the finished plugin is simple, but you have to install it first. You will only need the vsix file generated while packaging the extension with webpack.

Figure 24: the generated vsix file.

After acquiring the file, you need to open the extension tab in your editor. Next, open the menu with the three dots and choose "Install from VSIX".

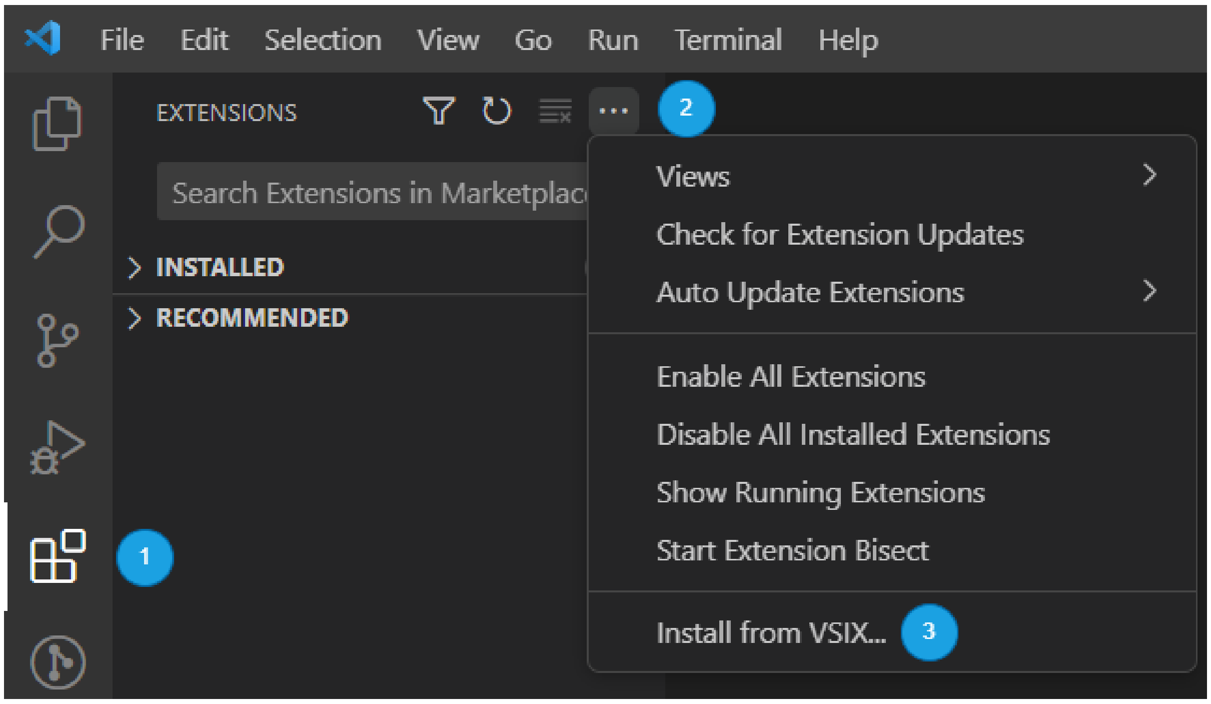


Figure 25: the steps to install the plugin in the editor.

This opens the standard file explorer of your system. Finally, navigate to the vsix file in the explorer and choose it for installation. Afterwards, the extension should be active whenever there are G-Code files in the active editor.

9 Conclusions and Future Work

In conclusion, the Siemens G-Code language was far more extensive than expected, with approximately 400 Productions and 5000 Tokens. For reference, the 2023 language specification for C# has 59 Productions and around 500 Tokens. [1] Therefore, we had to overcome challenges along the way, which resulted in a lot of lost time and a more or less favourable outcome.

Some planned components still need to be fully implemented for time reasons. The next step would have been to finish encoding the events into JSON. Next, the JSON Message would be read and processed into tokens and ranges. The tokens would be used to map the affiliated ranges to a DecoratorOption, which would then handle the appearance of said text. Furthermore, since Coco is also able to report SyntaxErrors, it would have been a useful tool to be able to display these errors with a wavy red underline and a message describing the error by also using a similar idea with DecoratorOptions or even displaying a pop-up window.

Despite the setbacks of not being able to implement the mentioned features on time, this does provide an opportunity to continue the work in the future and enhance the plugin, which, regardless of these unfortunate circumstances, is already a helpful tool to support developers in their efforts of writing G-Code.

List of Figures

References

[1] Microsoft. *Grammar - C# language specification | Microsoft Learn.* URL: https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/grammar. (accessed: 05.09.2023).

[2] Microsoft. *Visual Studio Code Extension API.* URL: https://code.visualstudio.com/api/references/vscode-api. (accessed: 07.08.2023).

[3] Regex101. *Regular Expressions 101.* URL: https://regex101.com/. (accessed: 28.08.2023).

[4] Siemens. *NC-Programmierung.* URL: https://cache.industry.siemens.com/dl/dl-media/562/109783562/att_1038101/v1/139197625099/de-DE/index.html#treeId=76638a9cdc41699295df2f06036599a4. (accessed: 07.08.2023).

[5] Tabnine Team. *Top 40+ VSCode Extensions for Developers in 2022.* URL: https://www.tabnine.com/blog/top-vscode-extensions/#:~:text=The%20thing%20that%20makes%20VS,overwhelming%20at%20the%20same%20time.. (accessed: 28.08.2023).

[6] Wikimedia. *Extended Bachus Naur Form - Wikipedia.* URL: https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form#Conventions. (accessed: 28.08.2023).

[7] Wikimedia. *G-Code - Wikipedia.* URL: https://en.wikipedia.org/wiki/G-code. (accessed: 28.08.2023).

[8] yeoman. *yeoman.* URL: https://yeoman.io/. (accessed: 16.08.2023).

Statutory Declaration