1. Introduction

a. Abstract

The F Prime Prime language, stylized as F`` and FPP, is a modeling language developed for the open source F Prime (F`) flight software framework. Currently, there is no integrated development environment support for the FPP language and therefore, no good way of editing the code. Most languages have supported text editors that provide functionalities such as keyword highlighting and grammar checks which make code editing easier and more intuitive. One of the popular text editors is Visual Studio Code and it has a marketplace for different extensions including language support. The goal of this project is to create a Visual Studio Code extension for FPP that will enhance the text editing experience and provide support for the language. This extension will also be available in the Visual Studio Code Marketplace.

b. Executive Summary

The problem posed for this project involves code editing for the FPP language. As stated before, there is currently no good way of editing code for this language because there are no text editing supports available. Without certain features, writing a large amount of code can result in issues regarding readability and error checking. In order to overcome this issue, the project will be centered around the creation of an extension for Visual Studio Code that will provide functionalities to support the FPP language.

The features that the project will aim at are syntax highlighting, tokenizing and semantic highlighting, and an IntelliSense-like feature for the text editor. The syntax highlighting feature will display the text in different colors based on the type of keywords that are specified in the language. This includes reserved words, symbols, identifiers, and comments. Semantic highlighting supplements this by coloring text by the meaning of the program and not just the fixed keywords of the language. In addition to the highlighting, an IntelliSense-like feature would include code completion and parameter information.

The highlighting features will allow for more readability of the text and provide context by showing the structure of the code. These will also aid in finding errors in the program and give the user more control over their text editing. Semantic highlighting, in particular, will contribute to a better understanding of the code's logic. The IntelliSense feature will increase efficiency in editing code as it will promote less keyboard input. It will also increase accuracy by decreasing the instances of mistyped entries.

2. Motivation

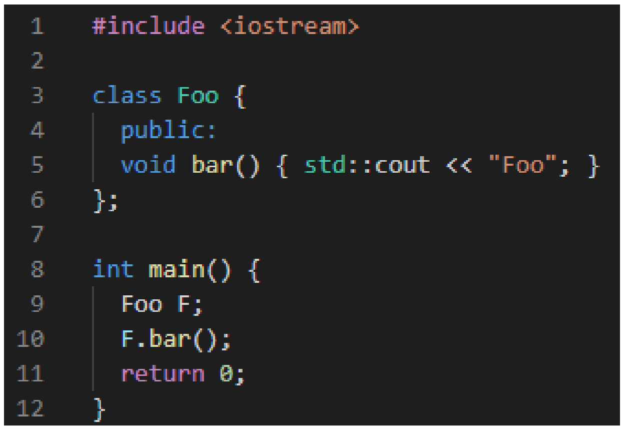
Microsoft's Visual Studio Code is inherently a text editor, however, its various extensions allow it to perform at the level of an Integrated Development Environment (IDE). The Extensions Marketplace allows users to browse and install publicly available extensions, many of which are published by Microsoft. Extensions can serve a variety of purposes but are primarily created to improve code writing and readability.

a. Competitive Analysis

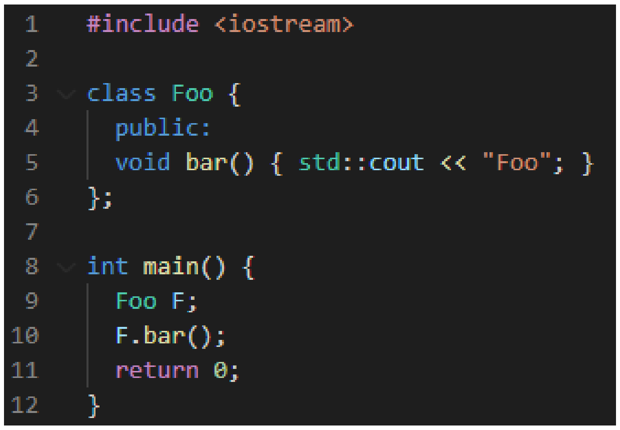
With the goal of this project to create an extension for FPP that will enhance the text editing experience and provide support for the language, we analyze extensions that serve a similar purpose. Three extensions we analyze are C# (powered by OmniSharp) by Microsoft, C/C++ by Microsoft, and Language Support for Java(TM) by Red Hat. There are many features consistent among all three of these extensions which are advantageous to their users.

Syntactic and Semantic Highlighting

Highlighting is the colorization or stylization of terms in programming, scripting, or markup source code. Highlighting can be performed syntactically where language-specific keywords and identifiers are highlighted based on their type and structure. However, semantic highlighting takes this process further by using a tokenization engine to highlight code based on its semantic use. In the example below, the left code snippet shows VS Code's built-in syntax highlighting and the right shows the C/C++ extension semantic highlighting. Notable changes can be seen on line 5 where the non-standard object 'cout' is highlighted and on line 9 where the class type 'Foo' is highlighted.



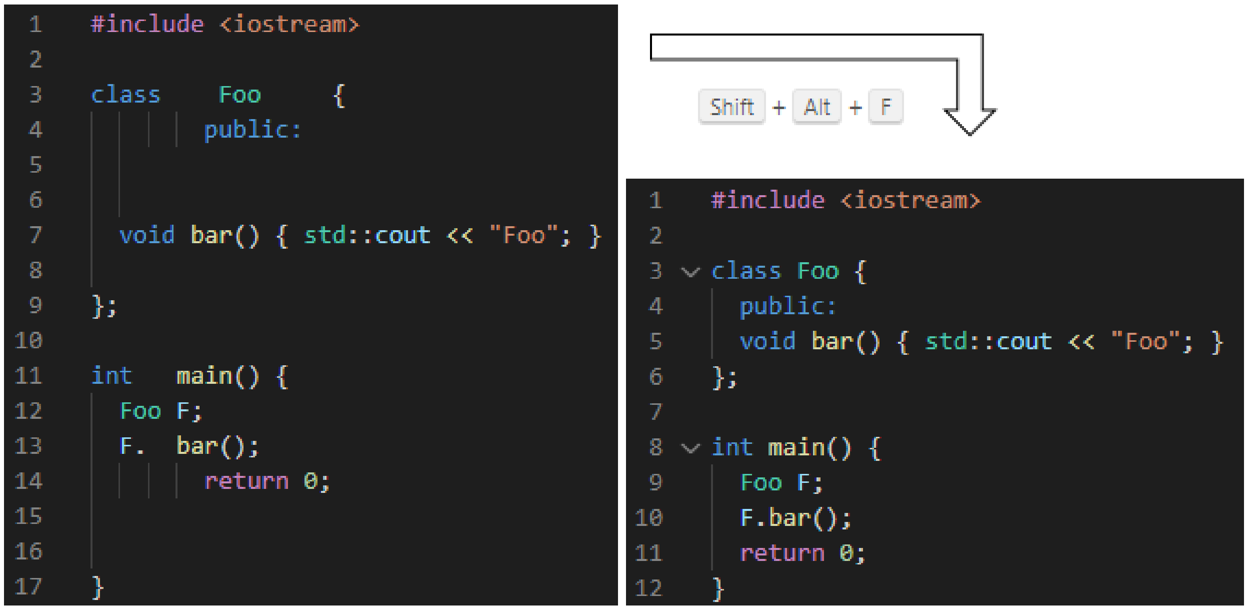
```
1   #include <iostream>
2
3   class Foo {
4     public:
5     void bar() { std::cout << "Foo"; }
6   };
7
8   int main() {
9     Foo F;
10    F.bar();
11    return 0;
12  }
```



```
1   #include <iostream>
2
3   class Foo {
4     public:
5     void bar() { std::cout << "Foo"; }
6   };
7
8   int main() {
9     Foo F;
10    F.bar();
11    return 0;
12  }
```

Auto-formatting

Auto-formatting speeds up the process of coding by reformatting the spacing and positioning of code. The desired format can often be specified, depending on the formatter that is used. It is important to note that the formatter does not alter the tokens (keywords, identifiers, operators, etc…) of the code, so only whitespace and line feeds will be altered. Additionally, in order to perform this function, the formatter must be able to parse the code requiring a tokenization engine and syntactically correct code. In the example below, the C/C++ extension reformats the left code snippet to the right in one command. The extension performs this by running the executable for the C/C++ formatter, 'clang-format'.



```
#include <iostream>

class    Foo     {
            public:


    void bar() { std::cout << "Foo"; }

};

int   main() {
    Foo F;
    F.  bar();
            return 0;


}
```

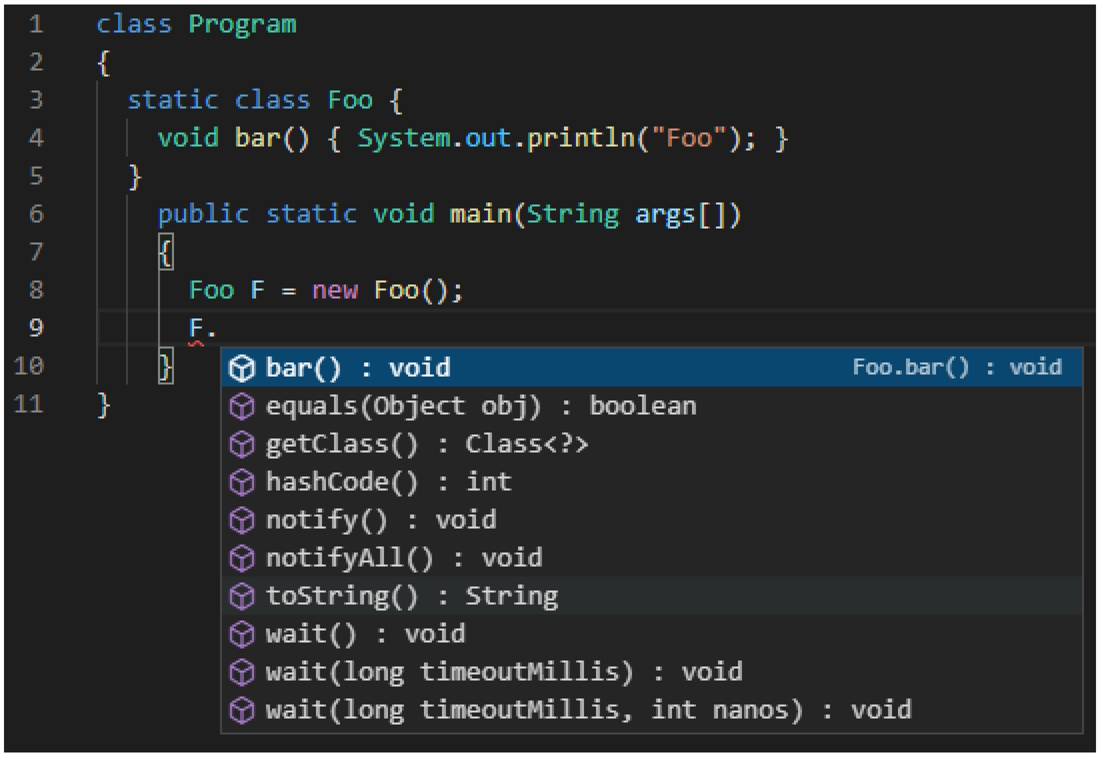
```
#include <iostream>

class Foo {
    public:
    void bar() { std::cout << "Foo"; }
};

int main() {
    Foo F;
    F.bar();
    return 0;
}
```

IntelliSense

IntelliSense is a general term but is typically understood as a code completion tool. Using a similar concept of tokenization necessary for other features, a language server analyzes the code to provide suggestions or auto-complete options within the editor. In the example below, the Java extension provides a list of methods upon typing the member operator '.' for the class 'F'. This list contains both the defined method contained on line 4 as well as the default methods contained in the Java object library.

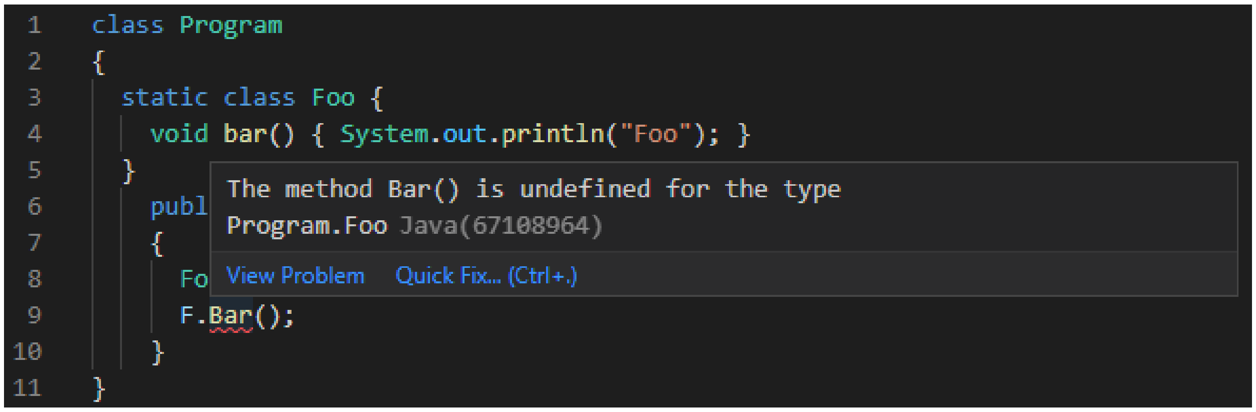


```java
class Program
{
  static class Foo {
    void bar() { System.out.println("Foo"); }
  }
    public static void main(String args[])
    {
      Foo F = new Foo();
      F.
    }
}
```

```
bar() : void                                   Foo.bar() : void
equals(Object obj) : boolean
getClass() : Class<?>
hashCode() : int
notify() : void
notifyAll() : void
toString() : String
wait() : void
wait(long timeoutMillis) : void
wait(long timeoutMillis, int nanos) : void
```

Linting

Linting is the identification of errors and bugs through code analysis performed by a linter without the need for compilation or code execution. Since there are many kinds of errors possible in programming languages, linters can vary in their complexity. In the below example, the linter built-in to the Java extension displays a red squiggly line under the ‘Bar’ function call. A mouse-over displays that the function is undefined. With this tool, we can quickly identify there was a casing error in the code.

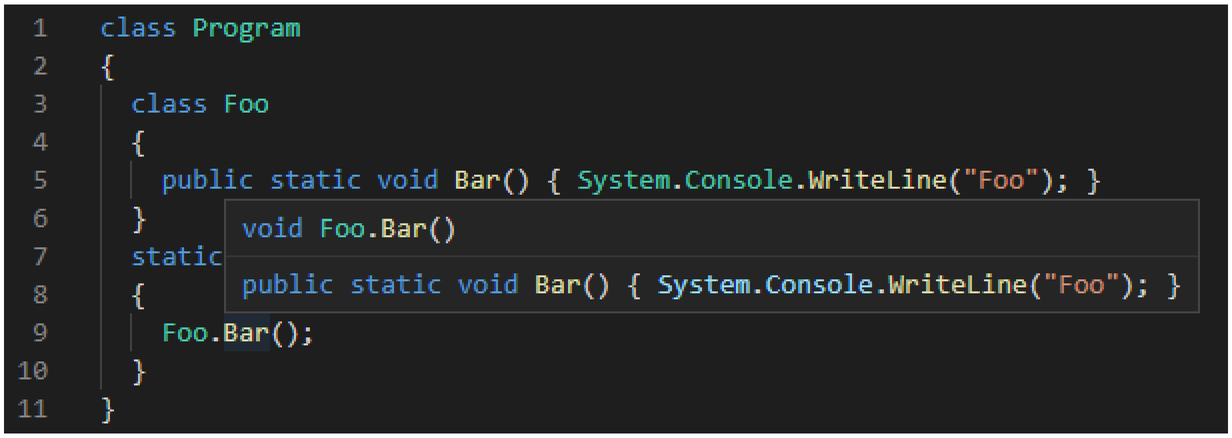


```java
class Program
{
  static class Foo {
    void bar() { System.out.println("Foo"); }
  }
    publ
    {
      Fo
      F.Bar();
    }
}
```

```
The method Bar() is undefined for the type
Program.Foo Java(67108964)
```

Code Referencing

Code referencing allows the user to quickly view or go to an object, function, or variable's declaration or definition. This provides convenience especially when working with foreign libraries or codebases. In the below example, mousing over 'Bar' while using the C# extension displays the function declaration. Additionally, mousing over and holding the control key displays the function definition.

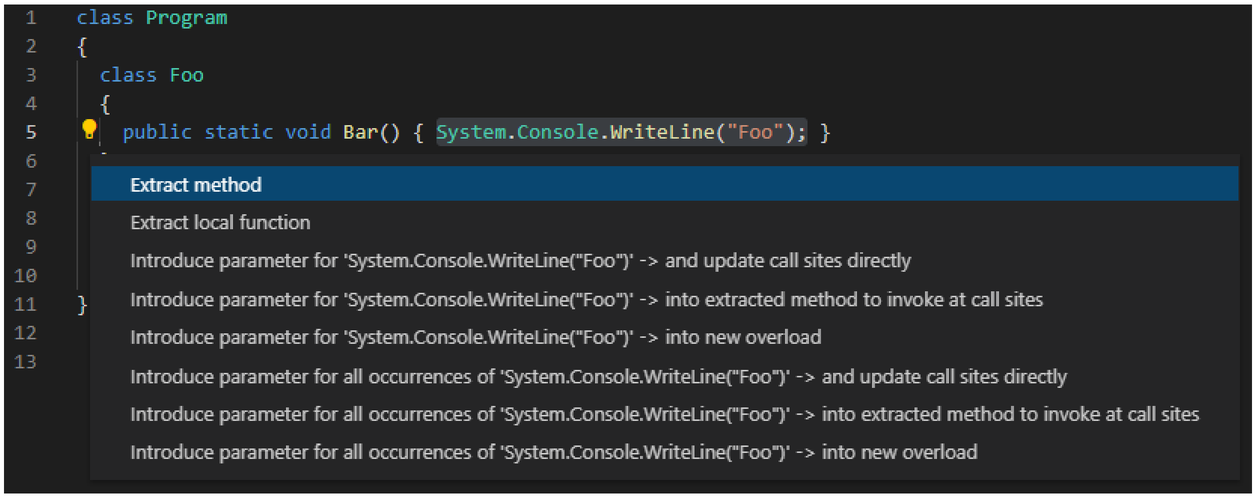


```
class Program
{
  class Foo
  {
    public static void Bar() { System.Console.WriteLine("Foo"); }
  }
  static
  {
    Foo.Bar();
  }
}
```

```
void Foo.Bar()
public static void Bar() { System.Console.WriteLine("Foo"); }
```

Code Refactoring

Code refactoring allows for the optimization or modification of code without changing its behavior. There are various implementations and uses for this depending on the language. In most cases it saves time and prevents errors when rewriting code. In the below example, selecting a target area and clicking the light bulb symbol lists potential changes which the C# extension will perform for you.



```
class Program
{
  class Foo
  {
    public static void Bar() { System.Console.WriteLine("Foo"); }

}
```

Extract method
Extract local function
Introduce parameter for 'System.Console.WriteLine("Foo")' -> and update call sites directly
Introduce parameter for 'System.Console.WriteLine("Foo")' -> into extracted method to invoke at call sites
Introduce parameter for 'System.Console.WriteLine("Foo")' -> into new overload
Introduce parameter for all occurrences of 'System.Console.WriteLine("Foo")' -> and update call sites directly
Introduce parameter for all occurrences of 'System.Console.WriteLine("Foo")' -> into extracted method to invoke at call sites
Introduce parameter for all occurrences of 'System.Console.WriteLine("Foo")' -> into new overload

This suite of features is common in most IDEs. All of these features serve to aid the user in producing more readable and optimized code with fewer errors. More features could be implemented to further reach this goal, however such features raise concerns of being too situational or shifting the balance of control a user has within the editor.

b. Journal Articles

José Franzim Miranda, D., Ferreira, M., Kucinskis, F., & McComas, D. (2019). A Comparative Survey on Flight Software Frameworks for 'New Space' Nanosatellite Missions. *Journal of Aerospace Technology and Management*. https://doi.org/10.5028/jatm.v11.1081

The article focuses on Flight Software Frameworks (FSW) which is a ground software used for nanosatellite missions. This involves virtual control of a spacecraft which in order to "safely" control such it must meet 14 selected criterias (pg.11). Furthermore, this article provides a comparison of other software and how those aid and or complement the FSW. The authors go into detail about the projects the FSW has launched and the development as it concerns what is needed of the software in order to be considered "successful" for missions (list of reference in pg. 3). Ultimately, the FSW framework is compared to about 6 software with the aforementioned (14) requirements and due to NASA cFS being similar to F Prime, F Prime ends up being taken into consideration.

As it relates to JPL F Prime, we can look at the relevance of the requirements needed for FSW as the JPL F prime language would be used for this same purpose. It is also mentioned how F prime components such as the fixed input and output nodes or static topology plays a role in considering if being used worthwhile for any nanosatellite mission project. Whether FSW uses F prime does not have as much relevance, but what is more relevant are the advances we can make knowing the important factors and componets which are looked into during these nanosatellite missions. By focusing on highlight, tokenization, and intellisense we are focusing on 5 of the components.

Watney, G., Levison, J., Bocchino, R., Canham, T., & Reder, L. (2018, August 9). *DigitalCommons@USU | Utah State University Research*. Https://Core.Ac.Uk/Download/Pdf/220136003.Pdf. Retrieved February 9, 2022, from https://digitalcommons.usu.edu/

Originally, F Prime was developed at JPL with the purpose of aiding and advancing the development of flight software for space missions at a smaller scale. With reviews and changes to the F Prime we now see it has more potential than expected and continued development is expected to meet the required criteria. During the introduction we not only see the purpose of F Prime but also the difficulties it may face as a method of virtual communication. It is important to note the aforementioned FSW is now being modeled by F Prime which is an important component for the project and the advancement of F Prime as it relates to software architecture.

Through this article we were able to find not only the relevance of F Prime as it relates to the projects involved but also components of such that we can expand on. It is important to note that we are able to find the port types and kinds and what type of data can be sent through the ports. This is important as we attempt to create our own ports in VS Code. The article also references FPP as it relates to F Prime and the advancements made and the future vision. The article highlights the importance of having a clean readable format. As we attempt to do highlight, tokenization, and intellisense it is imperative to maintain such format in our code.

Coppin, A. (2016). Project Bibliographies: Tracking the Expansion of Knowledge Using JPL Project Publications. *Issues in Science and Technology Librarianship*. https://doi.org/10.5062/F4XD0ZP5

Project Bibliographies is particularly important as it outlines the JPL Projects and how those are tracked as well as relevance of documenting. It goes into detail on how documentation is used in a wider range when it comes to involvement, research, and success. The importance of such requirements needs to be addressed in order to minimize cost and maximize the amount of work spent in each part/fragment of the JPL project. Due to the privacy of the JPL project not all of the advancements can be shared; however, the majority are outlined in such publications presented for better comprehensive reporting.

We found this article helpful as we work together with the JPL F prime staff. One of those requirements was the importance of having an outline of how we plan to achieve our goals with the project as there is a high probability of not only our group will be working on this project but to provide an understanding of the actions being taken and therefore someone else being able to replicate the same and continue the already work which is to be done. Being able to provide an articulate and detailed outline of the target our team is heading towards is important to be able to measure success and steps of the project. Such reporting can be used not just by our teams but also by the involved members of JPL F Prime.

3. Requirements

a. High-Level Description:

As part of the standard Language extension, keyword highlighting will be needed. We also are accounting for the addition of high-level syntactic (based on wording) or semantic (based on word meaning) highlighting as well. To achieve semantic highlight, however, an additional method of tokenizing the language F Prime Prime will be needed as well. Once that's all done, developing IntelliSense for the language, and allowing the project to be available on VS Code's marketplace, will be our final goals.

b. Functional requirements description:

ID	**Title**	**Requirement Description**	**Date**	**Done**
F1	Syntax Highlighting	Use regular language to find and highlight keywords and phrases, via VS Code's built-in tools.	2/05/2022	Yes
F2	Tokenizing	Develop a method for the computer to "understand" the FPP language, via tokenizing keywords.	4/03/2022	No
F3	Semantic Highlighting	Allow for highlighting based on tokens, rather than just by keywords.	4/20/2022	No
F4	IntelliSense	Develop a method of noticing syntax/semantic errors, as well as give suggestions on what the client should type next.	5/02/2022	No
F5	VS Code Marketplace Publishing	Publish the extension to VS Code's marketplace, so that anyone may access and use the extension.	5/03/2022	No

c. Functional requirements description:

1.) F1 - Syntax Highlighting:

All keywords and phrases specific to FPP will be uniquely highlighted. Syntax highlighting colors specific text based on the languages' lexical rules. The color scheme will be visible regardless of the current user's color theme.

2.) F2 - Tokenizing:
Along with proper syntax highlighting for FPP, the extension will have tokenization support. Tokenization involves breaking apart text to classify what each text segment is, and is given a token type. Tokenization will involve creating a grammar for FPP.

3.) F3 - Semantic Highlighting:
Semantic highlighting, or semantic tokenization within VS Code, to provide additional token information based on the language server's knowledge on how to resolve symbols in any given context of a project. Within the editor, the extension will apply highlighting to tokens via their semantic context in the client's program.

4.) F4 - IntelliSense:
IntelliSense within VS Code allows for recommended code completion, hover info, and function information so that code can be written more efficiently. FPP IntelliSense support will function similarly to standard IntelliSense on languages already supported by VS Code.

5.) F5 - VS Code Marketplace Publishing:
The extension will be made publicly available via the Visual Studio Code Marketplace, where users can easily install and use the application, as well as easily push updates and changes if needed.

d. Non-functional requirements table:

ID	**Title**	**Requirement Description**
NF1	Operating System	Any Operating System that VS Code supports.

NF2	Programming Language	The project will be written in TypeScript and JavaScript.
NF3	Host System	The extension will be made available on the public VS Code Extension Marketplace.

e. Non-functional requirements description

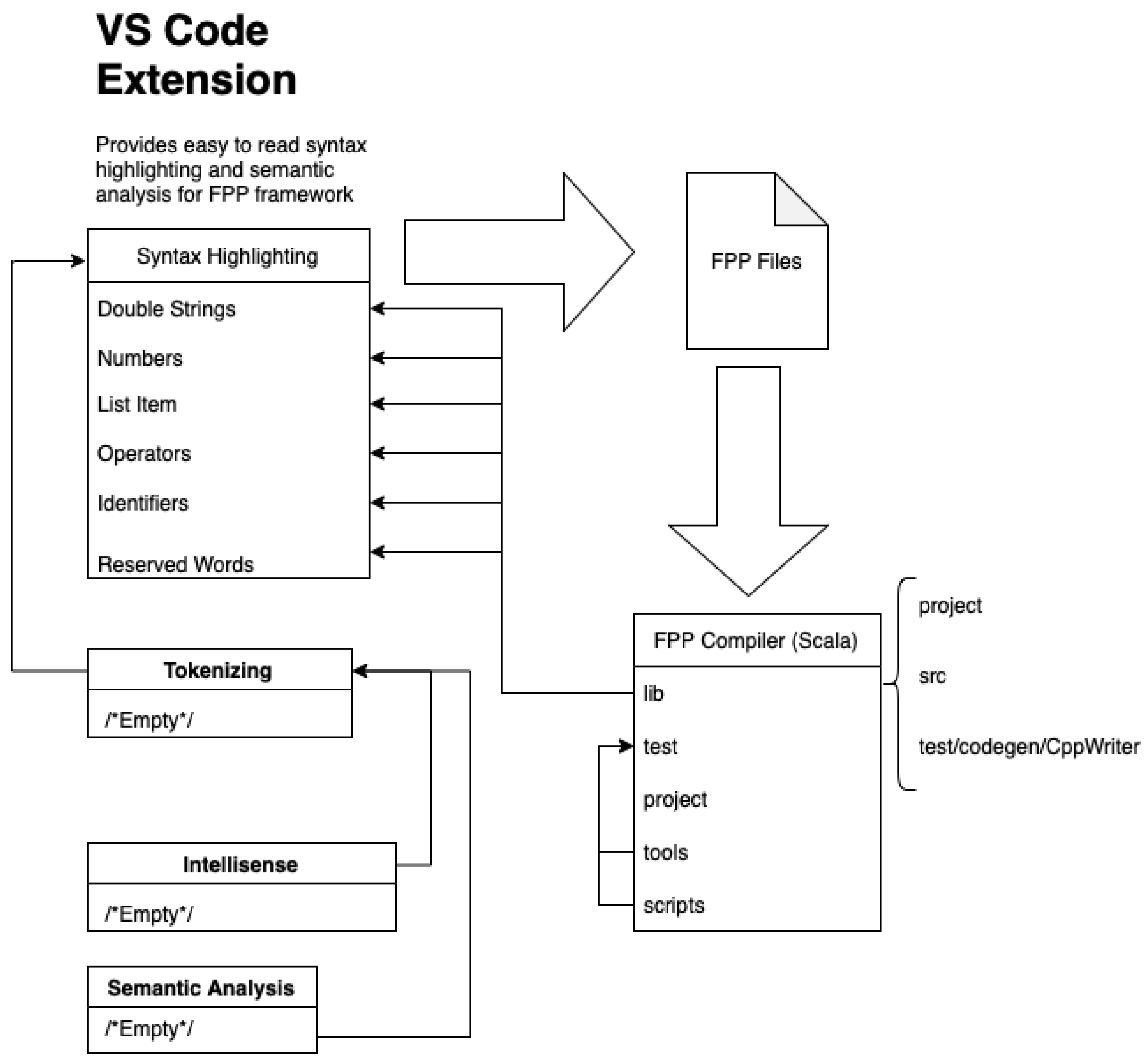
1.) NF1 - Operating System:
VS Code is supported on the following platforms:
- OS X El Capitan (10.11+)
- Windows 7 (with .NET Framework 4.5.2), 8.0, 8.1, 10, and 11 (32-bit and 64-bit)
- Linux (Debian): Ubuntu Desktop 16.04, Debian 9
- Linux (Red Hat): Red Hat Enterprise Linux 7, CentOS 8, Fedora 24

As such, the extension will also be supported on these platforms.

2.) NF2 - Programming Language:
TypeScript is a programming language developed by Microsoft, and is used as the main language to develop Visual Studio Code extensions. TypeScript is a superset of JavaScript, as such, the output from the compile process is plain JavaScript. Since TypeScript compiles to JavaScript, it provides support for many platforms and operating systems.

3.) NF3 - Host System:
Visual Studio Code has a built-in Extension Marketplace, where users can personalize their VS Code development environment to better suit their workflow, and extension developers have a centralized place to easily publish their extensions. Extensions serve to make VS Code users to personalize their development, whether it be having support for a language or compiler that isn't initially supported on VS Code, or change how their own VS Code looks and feels with custom color theming.

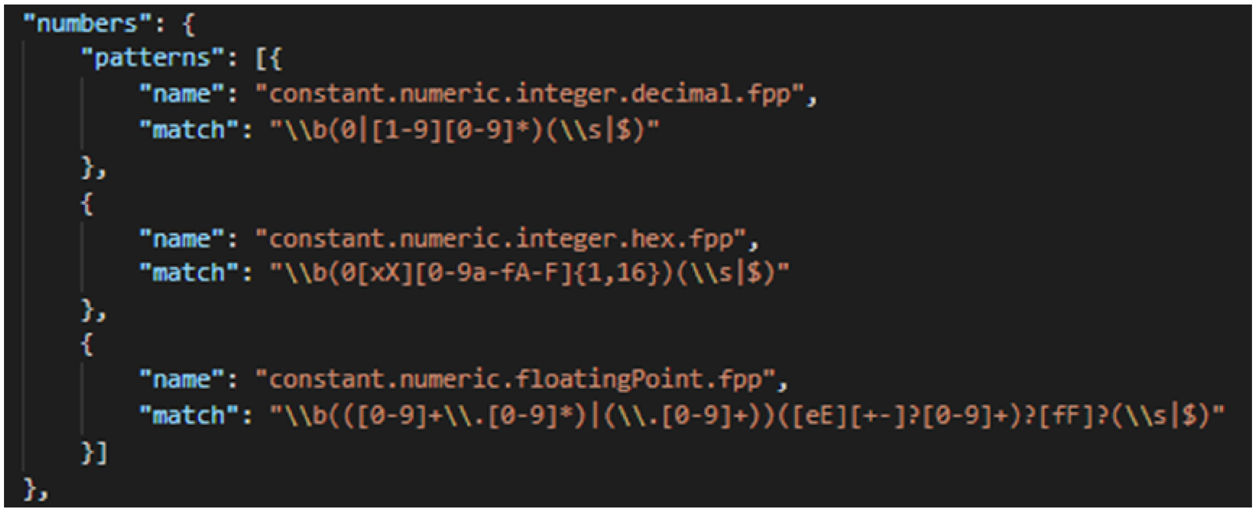
4. Design Document



The .fpp files are first read in by the scala based compiler. We then highlight the syntax by changing the color on the different keywords and reserved words using regular expressions that we made based on the FPP language. These same regular expressions used in Syntax Highlighting will be used to handle tokenization by assigning an English meaning to a piece of code. The tokenization will then be utilized in both Semantic Analysis and Intellisense.

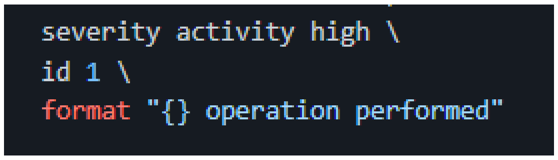
5. Prototype

The extension is a downloadable package that runs in the background, that the user will not have to interact with unless they choose to uninstall.



```
"numbers": {
    "patterns": [{
        "name": "constant.numeric.integer.decimal.fpp",
        "match": "\\b(0|[1-9][0-9]*)(\\s|$)"
    },
    {
        "name": "constant.numeric.integer.hex.fpp",
        "match": "\\b(0[xX][0-9a-fA-F]{1,16})(\\s|$)"
    },
    {
        "name": "constant.numeric.floatingPoint.fpp",
        "match": "\\b(([0-9]+\\.[0-9]*)|(\\.[0-9]+))([eE][+-]?[0-9]+)?[fF]?(\\s|$)"
    }]
},
```

Here is an example of the current code we have for number syntax highlighting, identifying the kinds of symbols/characters that can be viewed. More characters, reserved words, and operators are identified in the json file that we are using to hold the identification.

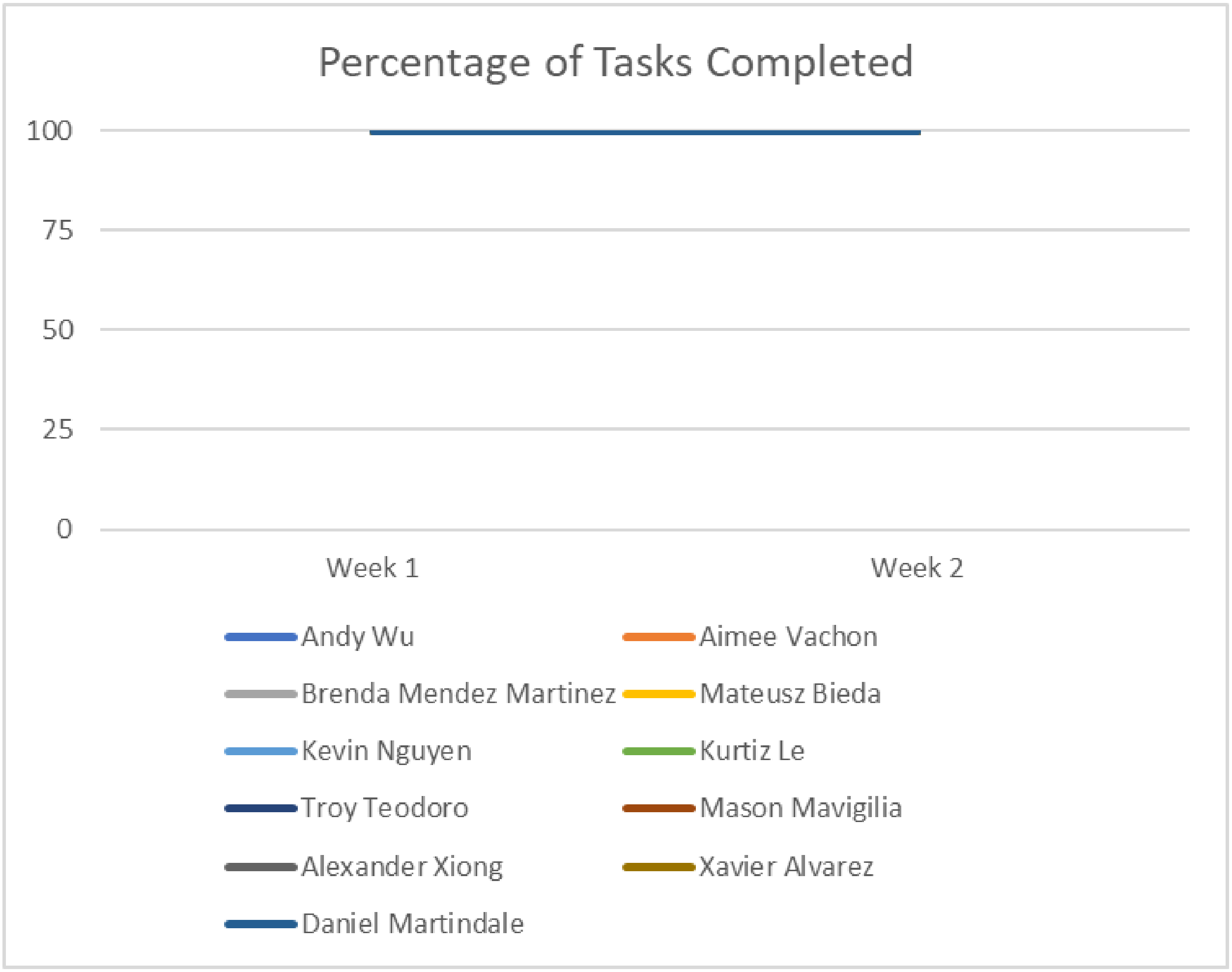


```
severity activity high \
id 1 \
format "{} operation performed"
```

In the development environment, we expect the code to look like this, where various parts are color coded and easily identifiable to the user.

6. Team Reporting

a. Team Tasks Chart



Andy Wu:

1. Week 1
   a. Investigate existing extensions (complete)
2. Week 2
   a. Begin Design Document for DD1 (complete)

Aimee Vachon:

1. Week 1
   a. Extension basics (get something small working) (complete)

    b. Commit 1: Added syntax highlighting for operators (complete)

2. Week 2

    a. Begin Introduction for DD1 (complete)

Brenda Mendez Martinez

1. Week 1

    a. Familiarize with F`` (complete)

2. Week 2

    a. Begin Journal Articles for DD1 (complete)

Mateusz Bieda

1. Week 1

    a. Study Typescript and connecting Scala (complete)

2. Week 2

    a. Begin Prototype for DD1 (complete)

Kevin Nguyen

1. Week 1

    a. Extension basics (get something small working) (complete)

2. Week 2

    a. Begin Conclusion for DD1 (complete)

Kurtiz Le

1. Week 1

    a. Extension basics (get something small working) (complete)

2. Week 2

    a. Begin Requirements for DD1 (complete)

Troy Teodoro

1. Week 1

    a. Understand F`` Compiler (complete)

2. Week 2

    a. Begin Design Document for DD1(complete)

Mason Maviglia

1. Week 1

    a. Extension basics (get something small working) (complete)

    b. Commit 1: Added all F`` reserved words to the syntax highlighter extension (complete)

2. Week 2

    a. Begin Team Reporting for DD1 (complete)

Alexander Xiong

1. Week 1

    a. Familiarize with Typescript, Scala, F` (complete)

    b. Commit 1: Added comments and pre/post Annotations syntax highlighting (complete)

2. Week 2

    a. Begin Team Reporting for DD1 (complete)

Xavier Alvarez

1. Week 1

    a. Extension basics (get something small working) (complete)

    b. Commit 1: Created a basic HelloWorld Highlighter Extension. (complete)

    c. Commit 2: Sorted the syntax lighting into easier to manage components. (complete)

    d. Commit 3: Added syntax highlighting for basic data types. (complete)

2. Week 2

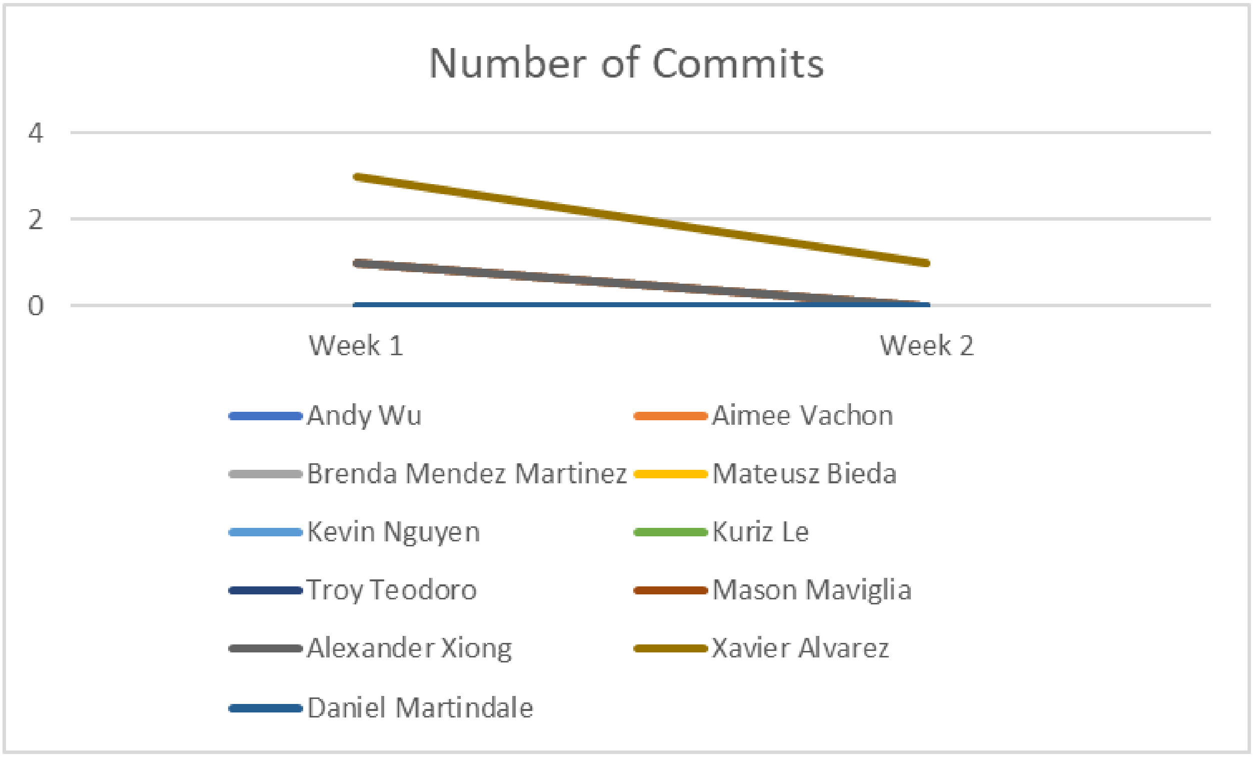
    a. Begin Requirements for DD1 (complete)

    b. Commit 4: Added variable highlighting, removed some things, and fixed some bugs. (complete)

Daniel Martindale

1. Week 1
    a. Familiarize with F`` (complete)
2. Week 2
    a. Begin Competitive Analysis for DD1 (complete)

b. Team commit chart



c. Success Report

Team Member	**Doc 2**	**Obtains Grade**	**Doc 3**	**Obtains Grade**	**Total Commits**	**Passes CS 472**
Andy Wu						

Aimee Vachon						
Brenda Mendez Martinez						
Mateusz Bieda						
Kevin Nguyen						
Kurtiz Le						
Troy Teodoro						
Mason Maviglia						
Alexander Xiong						
Xavier Alvarez						
Daniel Martindale						

7. Conclusion

Our team has begun our project of developing a Visual Studio Code extension for F’’ to use for the open source F Prime flight software framework developed by NASA’s JPL team. As requested by the clients, the extension will contain features that will assist with the coding process such as syntax highlighting, tokenizing and semantic highlighting, and an IntelliSense-like feature. The team has done some background research and has familiarized themselves with the F’’ language. Out of these main goals, the team has completed a prototype version of syntax highlighting. When the project is completed, it is planned to be published as open-source and directly downloadable from Visual Studio Code’s marketplace.