

**PROJECT REPORT**

Syntax Dictionary for C#

**Author: Mikki Katyal and Ajay Kumar**

**Presented To:**

**Sazzadur Rahaman**

**Assistant Professor**

**University of Arizona**

**TABLE OF CONTENTS**

Contents

[1. Executive Summary 3](#_Toc185009998)

[2. Project Summary 4](#_Toc185009999)

[3. Objectives and Goals 5](#_Toc185010000)

[4. System Design and Architecture 5](#_Toc185010001)

[5. Features and Functionality 6](#_Toc185010002)

[6. Challenges and Solutions 6](#_Toc185010003)

[7. Testing and Validation 6](#_Toc185010004)

[8. Conclusion 7](#_Toc185010005)

[9. Future work 7](#_Toc185010006)

## Executive Summary

Automation of software writing process is a need of the time as requirement of new software applications is growing day by day. Having a dictionary, syntax and expressions that can be used along with variables having values and syntax string to create a customer made source code based on the given parameters is an important component of code generation. This dictionary can be a running flat that shall be updated as and when desired to accommodate new complex expressions in the source code. This will enable the software developers to keep their focus on the business of the customer, with code generation applications available to aid them in the changing business requirements of the customer.

Syntax Dictionary is a key component of a code generation process, as it brings the agility to update and store the most used syntax and the most complex expressions to be written in a source code file that is custom made for a particular project.

It contributes to the structured process of code writing and minimizes the randomness in code structures that is an inherited part of manual code writing.

The dictionary is part of a design for an “All Code” Code generator, unlike a “Low Code” or “No Code” platform. This enables the developer to access the source code just like any other manually written code to perform all required operations like manual code review, dry runs and debugging.

With a flat file format, the dictionary can be accessed manually for review, and it contributes to speed of reading, writing and organizing the data in the dictionary. The format is kept simple so that it can be replicated for any other language going forward.

## Project Summary

This project automates the creation of a syntax and expression dictionary for C#, addressing key challenges faced by software developers such as development speed, standardization, readability, and ease of maintenance. The project's main objective is to build a C# class library capable of reading source code files, identifying existing and new elements in the code, and updating a dictionary accordingly.

The dictionary is updated continuously as new expressions are found, storing complex and frequently used code elements. By generating a dictionary of syntactical elements, developers can streamline the process of code generation, making it easier to create custom source code.

Key features of the project include:

* **Reading C# Source Code:** The tool reads source code files line-by-line and processes each element.
* **Identifying Known and New Elements:** The tool compares each word in the source code with existing dictionary entries and flags new elements for inclusion.
* **Updating the Dictionary:** New elements are added to the dictionary after being validated based on predefined rules and C# keywords.
* **Creating/Updating a Flat File Dictionary:** The dictionary is stored in a flat file format, making it easy for developers to access, review, and modify.

The dictionary will be used by a larger code generation application that aims to simplify the software development process by automating code writing, while still allowing developers to manually review and adjust the generated code. This solution will significantly improve the development process by increasing efficiency, reducing errors, and improving consistency.

In the future, the dictionary system can be extended to support multiple programming languages, further enhancing the code generation capabilities of the tool. File format of the dictionary will allow for easy access and manual review, ensuring that the tool is user-friendly for developers.

By the end of the project, the team aims to deliver a robust library capable of simplifying and standardizing the code generation process for C# developers, with potential applications extending to other programming languages in the future.

## Objectives and Goals

* **Primary Objective:** Automate the creation and maintenance of a syntax and expression dictionary for C#.
* **Secondary Goals:**
  + - Improve the efficiency and speed of code generation.
    - Ensure standardization and readability of generated code.
    - Make the dictionary extensible to other programming languages.
    - Allow manual review and updating of the dictionary as required.

## System Design and Architecture

The core of the project involves building a C# class library designed to:

* Read C# Source code and break it down into individual elements
* Read the existing dictionary of syntax elements and expressions.
* Compare the elements of source code with elements in existing dictionary.
* Flag new elements ( those not found in the existing dictionary ) from source code.
* Elements matching the business rules will be labeled with automation.
* Remaining new elements can be labeled by the user.
* Final list of new labeled elements is ready.
* User will use "Write Dictionary" button to append the new labeled elements in the existing dictionary.

The architecture of the system follows a modular design:

* **Source Code Reader:** Reads C# code and extracts the individual components (variables, expressions, etc.).
* **Dictionary Validator:** Compares each component with the existing dictionary to determine if it is new or known.
* **Dictionary Updater:** Adds new elements to the dictionary and ensures its accuracy.
* **File Storage:** Stores the dictionary in a flat file, enabling manual access and updates.

**User Interface:** -

`A screenshot of a computer

Description automatically generated

Step 1: Select Existing dictionary to read.

* Code element from dictionary will be listed in “Existing Dictionary”

Step 2: Select Code file to read

* Element of source code found in “Existing Dictionary” will be listed into “Source Code-Word found in Dictionary”
* Element of source code **NOT** found in Dictionary will be listed into “Source Code-New Words”
* Elements listed in “Source Code-New Word” will be parsed through business rules and will be labeled automatically. Such elements will be listed in “New Words – Add to Dictionary”
* Elements in “Source Code – New Word” can be labeled manually. User will have to select one word from this list and a relevant label from “Labels”, Press the “Label” button to label the selected word and insert into the list “New Word – Add to Dictionary”. The unique label is create as a constant.

Sep 3:

* Once the set of new words (labeled as Constants) is final. User can click the button “Write Dictionary” to select the Dictionary file that will be appended with the constants from “New words – Add to Dictionary”

Sep 4:

* Users can reopen the latest Dictionary to validate and view the update of Code elements in latest dictionary.

Dictionary format: (Output)

A screenshot of a computer program

Description automatically generated



Sample Source Code of C#: (Input)

A screenshot of a computer

Description automatically generated



## Features and Functionality

The main features of the system include:

* C# Source Code Parsing: The library parses the provided C# source code and breaks it into individual components.
* Known vs New Element Identification: Each component is checked against the dictionary to identify new expressions that need to be added.
* Dictionary Management: The system ensures that all identified new expressions are added to the dictionary in an organized manner, supporting efficient code generation in future projects.
* Flat File Output: The dictionary is saved in a flat file, making it easy to access and modify by developers.

## Challenges and Solutions

During the development of this project, several challenges were encountered:

* **Handling Complex Expressions:** Managing complex C# expressions and ensuring they are correctly parsed and added to the dictionary was challenging. A structured approach to parsing and rule-based validation was employed to address this issue.
* **Scalability:** As the dictionary grows with each added element, ensuring it remains manageable and efficient was a key concern. A flat file format was chosen for its simplicity and ease of access.

The solutions implemented successfully addressed these challenges and resulted in a working prototype capable of automating the dictionary creation process.

## Testing and Validation

The dictionary tool was tested using a variety of C# source code files to ensure it accurately identifies new elements and adds them to the dictionary. Testing included:

* **Unit Testing:** Verifying individual components of the tool.
* **Integration Testing:** Ensuring the entire system functions cohesively.
* **Manual Review:** Developers manually reviewed the generated dictionary to ensure accuracy and completeness.

The results confirmed the system's effectiveness in automating the process of dictionary creation, ensuring faster and more efficient code generation

## Conclusion

The project successfully automated the creation of a syntax and expression dictionary for C# code, enabling more efficient code generation. The system streamlines the process of adding and organizing code elements, reducing manual effort and improving consistency. The dictionary's flexibility allows it to be extended to other programming languages in the future, offering scalability and adaptability to various code generation applications.

The solution meets the growing need for automation in software development, contributing to faster, more efficient, and more maintainable code generation.

## Future work

Future enhancements to the system include:

* **Support for Multiple Languages:** Extending the dictionary to support additional programming languages beyond C#.
* **Integration with a Full Code Generator:** Linking the dictionary to a complete code generation framework that can generate full source code from defined parameters.
* **Improved User Interface:** Developing a user-friendly interface to facilitate easier management of the dictionary by developers.

The project has laid the foundation for a scalable and efficient code generation system that can contribute significantly to software development practices.