Software Requirements Specification (SRS)
Project: Macro Buddy

Team: Macro Buddy 2

Authors: Ryan Allen, Joseph Calles, Likhitha Eda, Tammy Liu, Nicholas Park

Customer: Software Developers

Instructor: Dr. James Daly

TABLE OF CONTENTS

INTRODUCTION 1

- Purpose 1
- Scope 1
- Definitions, Acronyms, and Abbreviations 2
- Organization 2

OVERALL DESCRIPTION 3

- Product Perspective 3
- Product Functions 4
- User Characteristics 5
- Constraints 5
- Assumptions and Dependencies 6
- Apportioning of Requirements 6

SPECIFIC REQUIREMENTS 7

MODELING REQUIREMENTS 8

- Use Case Diagrams 8
- Class Diagrams 12
- Sequence Diagram #1 16
- Sequence Diagram #2 17
- State Diagram 18

PROTOTYPE 19

- How to Run Prototype 22
- Sample Scenarios 23

REFERENCES 24

POINT OF CONTACT 25

1 Introduction

This SRS document is divided into seven subsections. In the first section we will discuss the purpose of the document, the scope of the product, and the definitions of key words, acronyms and abbreviations related to the product. In the second section we will talk about the product perspective which discusses more closely about the context of the product, the user interface, hardware interface, software interface, communication, operation, and constraints. Then this section will discuss the major product functions with diagrams to illustrate these functions. We briefly discuss the user expectations for the product as well as constraints, assumption, dependencies, and apportioning requirements.

We will also discuss specific requirements and modeling requirements in their subsections. The third section consists of the list of specific requirements we created for the product. The fourth section contains modeling requirements. This section contains the use case diagram, class diagram, and sequence diagrams with explanations. The fifth section will describe the functionality of the prototype. The sixth section contains references and finally the seventh section contains the point of contact information.

1.1 Purpose

The purpose of the SRS document is to explain the steps our team took to build the prototype for the software product. The document contains the overview of the product as well as the goals, the characteristics, requirements, and prototypes we designed for the product. This document is designed for students in the Software Engineering 1 course and for software developers interested in the planning and design of Macro Buddy 2. This document mimics the steps a software developing group will take to create a software product.

1.2 Scope

MacroBuddy2, is an application that records command line procedures and stores them to use in the future. Many times, when we want to run the same commands, it takes so much time and energy. Therefore, the MacroBuddy2 application will make the process of running the same commands more than once, easier, faster, and with ease to the user. The objective of MacroBuddy2 is to create a macro which will contain user given commands and automation script to run these commands.

The MacroBuddy2 will be able to record user given commands and create a script to store the commands to use to run later. This script will be stored in a location on their device chosen by the user. The MacroBuddy2 will display up to 50 macros on the user interface with a scrollable feature. The user will be able to select any macro they created and run it at any time they wish.

1.3 Definitions, acronyms, and abbreviations

Macro – A data structure that contains the user’s commands to be executed in sequence.
Macro Grid – A collection of macros displayed on the user interface organized as a grid, containing buttons and placeholders for the macros.
Terminal Emulator – The interface that the user will type their commands in, the user input will be used to create the Macros.
T.E. – Shorthand for Terminal Emulator.
User Interface (UI) – The interface which contains the Terminal Emulator and a series of buttons which the user can utilize to interact with the Emulator and their Macros.
MB2 – Shorthand for MacroBuddy2
Location – The file directory the user will save their macros.
Terminal Session – This is the terminal session starts when you open the Macro Buddy 2 application. This refers to the terminal that opens within the MB2 interface.

1.4 Organization

The rest of this document is organized as follows: Section 2 will contain the product perspective, product functions, constrains, user expectation, assumption, and dependencies. Section 3 will contain the software requirements. Section 4 will contain the product diagrams: use case, class, and sequence diagrams. Section 5 will contain a description of the prototype functionality. Section 6 contains references. Section 7 has the point of contact.

2 Overall Description

This section is divided into 6 subsections and we will discuss MacroBuddy2's context, structure, software functionalities, and more. In subsection 2.1 will discuss the product's system, software, hardware, user, and communication interfaces. Subsection 2.2 will discuss the main functions of the product. Subsection 2.3 will discuss what types of users we expect to use the product. Subsection 2.4 will discuss the legal constraints for the product. Subsection 2.5 will talk about what type of software environment will be needed to run the product. Finally, subsection 2.6 will discuss the requirements of the product that were out of scope for now and should be addressed in the future.

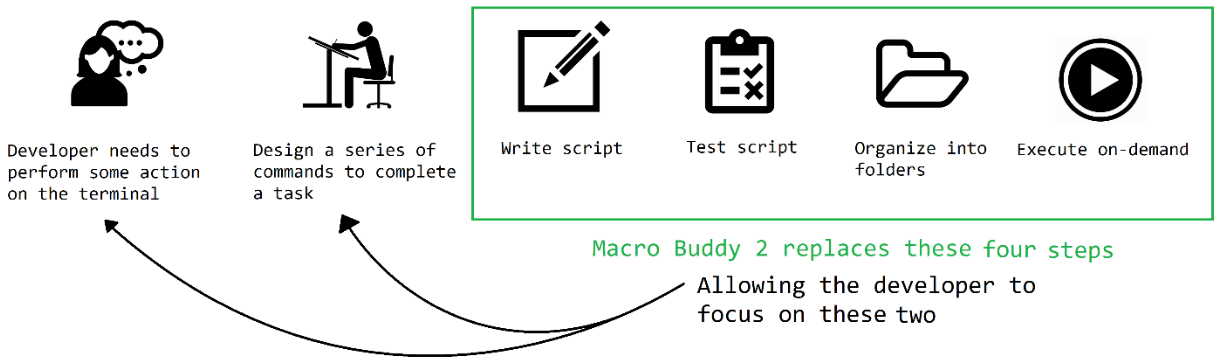
2.1 Product Perspective

The terminal command line interface has become a symbol of Computer Science, and even non-tech-savvy people recognize its design but view it with a sense of mystical wonder. Navigating a terminal and understanding its commands is an essential yet time-consuming part of software development. Macro Buddy 2 will help software developers save their terminal scripts and make them more accessible.

MB2 is meant to provide a layer of abstraction for terminal commands but does not entirely remove the need for a user to work in a terminal. Once macros are generated, the user can use MB2 as a high-level abstraction layer to easily reproduce the results obtained by previously written macros with a touch of a button that references macros stored in the file system.

The goal of MB2 is to quicken the development time of applications by giving developers a useful tool to handily recall and run terminal commands. It is meant to be a graphical user interface overlay for a system terminal; any system that uses MB2 should have such necessary prerequisites as a preinstalled terminal, user I/O capabilities, memory, and file manipulation.

MB2 works by recording user commands then parsing these commands and storing them. MB2 will ask the user for the location to save. MB2 can communicate with internal file systems through paths specified by the program itself so that the user does not need to worry about locating generated macros, thus acting as a shortcut and connection between the user and the internal file system of the computer.

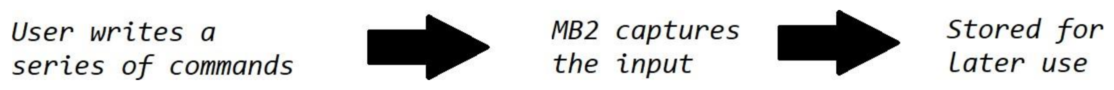

Developer needs to perform some action on the terminal
Design a series of commands to complete a task
Write script
Test script
Organize into folders
Execute on-demand
Macro Buddy 2 replaces these four steps
Allowing the developer to focus on these two

2.2 Product Functions

The primary function of MacroBuddy2 is to record a series of terminal commands imputed by the user, validate those commands, and then store the macro for later use. Storing the macros provides the user with the convenience needed to make the development process more efficient.

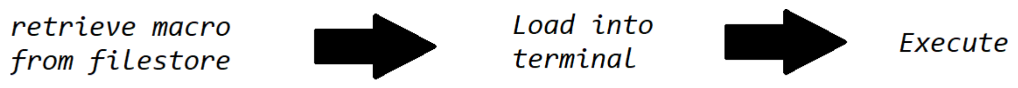
The two major functions of MB2 can be summarized in the following two illustrations which will be broken-down into more detail in section 4.

Function 1



Whereas users would normally spend the time either reentering the same commands into the terminal over and over, or spent hours developing a batch file, make file, or other type of command input file, MB2 affords the user the convenience of only needing to enter the commands one time without having to develop a script.

Function 2



This function streamlines the process of executing scripts. MB2 has a built-in "Macro grid" that is used to organize the macros in such a way as to make them easy to find by the user while navigating the MB2's user interface.

Besides these two functions, the user interface utilizes many helper functions which make the MB2 more user-friendly the work with. Such functions include optimized search algorithms, input processors, and code generators.

2.3 User Characteristics

While MB2 is targeted for use by software developers, anybody with an understanding of terminal commands can readily use this software. The user is expected to be aware of command line inputs, file navigation, and simple recording software. The software has mechanisms in place that try to prevent the misuse of macros, but this will not stop the misuse of the software within an environment where the user is expected to perform professionally. For example, MB2 will not prevent the user from running a macro multiple times in-a-row, even if it would cause problems. For this reason, the user should understand the potential dangers of using command line macros.

The user should be knowledgeable of command line scripting languages such as bash or shell script. Although MB2 has functions to correct user errors and functions to generate script files, they are not replacements for actual knowledge of scripting. Ignorance of this subject on the user's part could potentially lead to serious errors. As such, it is the user's responsibility to handle the MB2 software in a safe manner.

2.4 Constraints

The constraints for the MB2 are listed below:

- The MB2 software is constrained to work only within the environment provided by the hardware it is installed onto.
- MB2 will not attempt to make any external connections via the internet or other networking medium.
- MB2 will not attempt to access folders requiring passwords or admin privileges unless such is given by the user.
- MB2 will have a limited memory usage which it will use from your computer to store the macros. The MB2 shall not exceed this limit unless the user allows it.
- MB2 will only access folders in the file system within the project itself.
- MB2 will be able to run in the background with other apps running at the same time.

2.5 Assumptions and Dependencies

We are also assuming the user can download the application and has knowledge of file directory navigation. The user should be competent regarding terminal commands. Macro Buddy 2 can be used with MacOS (jar version) or with Windows (exe version) and supports the machine's command line functionality, the user is assumed to be using the correct version for their machine and is also assumed to be only using it to the extent it was designed, for example MacroBuddy does not function with other terminal emulators or command interfaces other than the machine's native default.

The user is required to have the Java Runtime Environment 15.0.1 installed. The user's machine is assumed to be operational and in possession of at least 50mb of free memory to run.

2.6 Apportioning of Requirements

Currently MacroBuddy2_v0.2 (the current build as of writing this) is capable of being used with MacOS (jar version) and Windows OS (exe version) and each version functions solely with the operating system's command line interface. MacroBuddy2 allows the user, via stripped down built-in terminal emulator, to record and store lists of commands to be executed in sequence (these are macros), both functioning and non-functioning and it is capable of loading and then executing those stored commands via quick button press. MacroBuddy2 keeps a running list of all Macros created and stored within MacroBuddy2's default directory but also allows the user to manually load a Macro from outside of this directory. The Terminal Emulator in its current state does not emulate terminal functionality aside from look, feel, accepting input, and inability to edit prior lines; given much more time something more akin to a true terminal emulator may have been possible.

3 Specific Requirements

1) The Macro Buddy should have a Terminal Emulator (T.E.) embedded in it.
    - 1.1) The T.E. should always be active whenever Macro Buddy is on.
    - 1.2) The T.E should recreate the look and feel of a Terminal
    - 1.3) The T.E. should be able to copy the user's commands and store them for generating a macro later in the program's life.

2) The user should be able to record a session from the Terminal Emulator.
    - 2.1) The user should be able to start the session and stop the session on the macro buddy using the GUI buttons.
    - 2.2) The interface for the macro buddy should mimic the appearance and functionality of a real terminal.
        - 2.2.1) The user should be able to type and edit commands such as on a real terminal.
        - 2.2.2) The T.E. should have access to current working directory in order that it may execute commands.
    - 2.3) Entries will be captured from the terminal and stored in a file which will be later accessible in the macro grid.

3) The Macro Buddy should be able to generate an Macro from user input.
    - 3.1) After the user stops recording, MB2 should be able to process the input in order to generate a macro file.
    - 3.2) MB2 should create a macro supported by the user's OS.
    - 3.3) Once input has been recorded, the macro buddy should be able to successfully generate a macro file within at most two seconds using a 2.6 GHz processor.

4) A file organization system dubbed as "Macro Grid" will be used to organize the macro files into an ordered list of macros that will be displayed on the GUI.
    - 4.1) The macro grid should contain all the macros recorded by the user up to at least fifty individual macros.
    - 4.2) As new macros are recorded, they should be added to placeholder slots on the Macro Grid GUI layout.

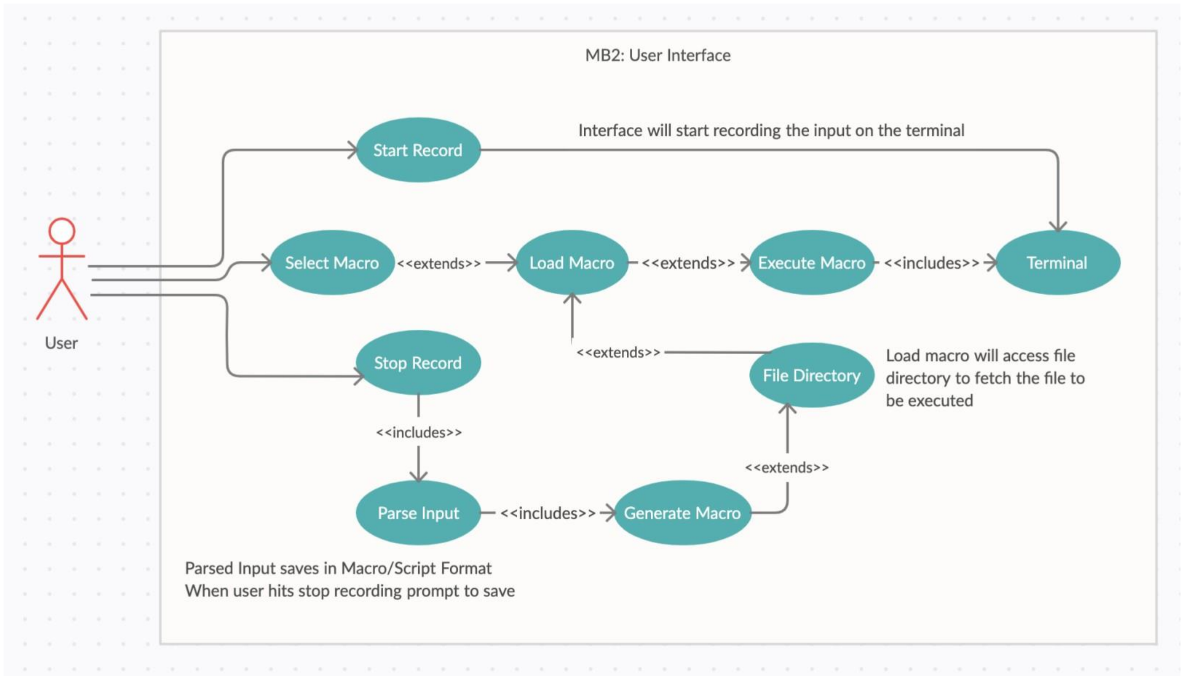
5) MacroBuddy2 should maintain the ability to track its generated Macro files.
    - 5.1) All macros by default, will be stored within the program's directory.
    - 5.2) The program shall allow the user to load Macros from outside of the default directory.
    - 5.3) The program's tracking system shall be consistent and loadable across multiple sessions.

4 Modeling Requirements

This section explains the modeling of the code in its concrete form; models are presented as illustrated diagrams. First, the use case diagram is presented to illustrate a map of the program and to draw the essential connections between function calls. Each use case is described in detail below the diagram. Second, the class diagram shows how each element relates to each other. Each element, along with their attributes and operations, are given a brief description below the diagram. Third, two sequence diagrams illustrate the flow of functions in order to conduct a single operation. Fourth, we use a state diagram to show the different states that the MB2 application can be in.

4.1 Use Case Diagram:

The use case Diagram illustrates how one actor, the user, will be able record commands, save the generated macro in a location they choose, and select a macro to run. In the diagram, the main operations of the product are shown as primary use cases and the rest of the use cases extend these primary use cases.


MB2: User Interface
Interface will start recording the input on the terminal
Start Record
User
Select Macro
<<extends>>
Load Macro
<<extends>>
Execute Macro
<<includes>>
Terminal
Stop Record
<<extends>>
File Directory
Load macro will access file directory to fetch the file to be executed
<<includes>>
<<extends>>
Parse Input
<<includes>>
Generate Macro
Parsed Input saves in Macro/Script Format
When user hits stop recording prompt to save

Use Case Name:	***Start Record***
Actors:	User
Description:	The User presses the Record button to start recording input from terminal.
Type:	Primary, Essential
Includes:	N/A
Extends:	N/A
Cross-refs:	Requirement 1, 3.1
Uses cases:	None

Use Case Name:	***Select Macro***
Actors:	User
Description:	The User selects a Macro to execute.
Type:	Primary
Includes:	N/A
Extends:	Load Macro
Cross-refs:	Requirement 1
Uses cases:	None

Use Case Name:	***Stop Record***
Actors:	User
Description:	When the user stops recording, input is parsed and saved
Type:	Primary
Includes:	Parse Input
Extends:	N/A
Cross-refs:	Requirement 3.1
Uses cases:	None

Use Case Name:	***Parse Input***
Actors:	N/A
Description:	The Parse Input function happens automatically when recording is stopped.
Type:	Secondary
Includes:	Generate Macro
Extends:	N/A
Cross-refs:	Requirement 3.1
Uses cases:	Stop Recording

Use Case Name:	***Generate Macro***
Actors:	N/A
Description:	Macro is generated after the input is parsed and created executable.
Type:	Secondary
Includes:	N/A
Extends:	File Directory
Cross-refs:	Requirement 5.1
Uses cases:	Parse Input

Use Case Name:	***File Directory***
Actors:	User
Description:	The file directory given by the user is accessed to save macros or load macros
Type:	Secondary
Includes:	Parse Input
Extends:	Load Macro
Cross-refs:	Requirement 6.1
Uses cases:	Generate Macro

Use Case Name:	***Load Macro***
Actors:	User
Description:	The user selected macro will load the file corresponding to it from the file directory.
Type:	Secondary
Includes:	N/A
Extends:	Generate Macro
Cross-refs:	
Uses cases:	Select Macro

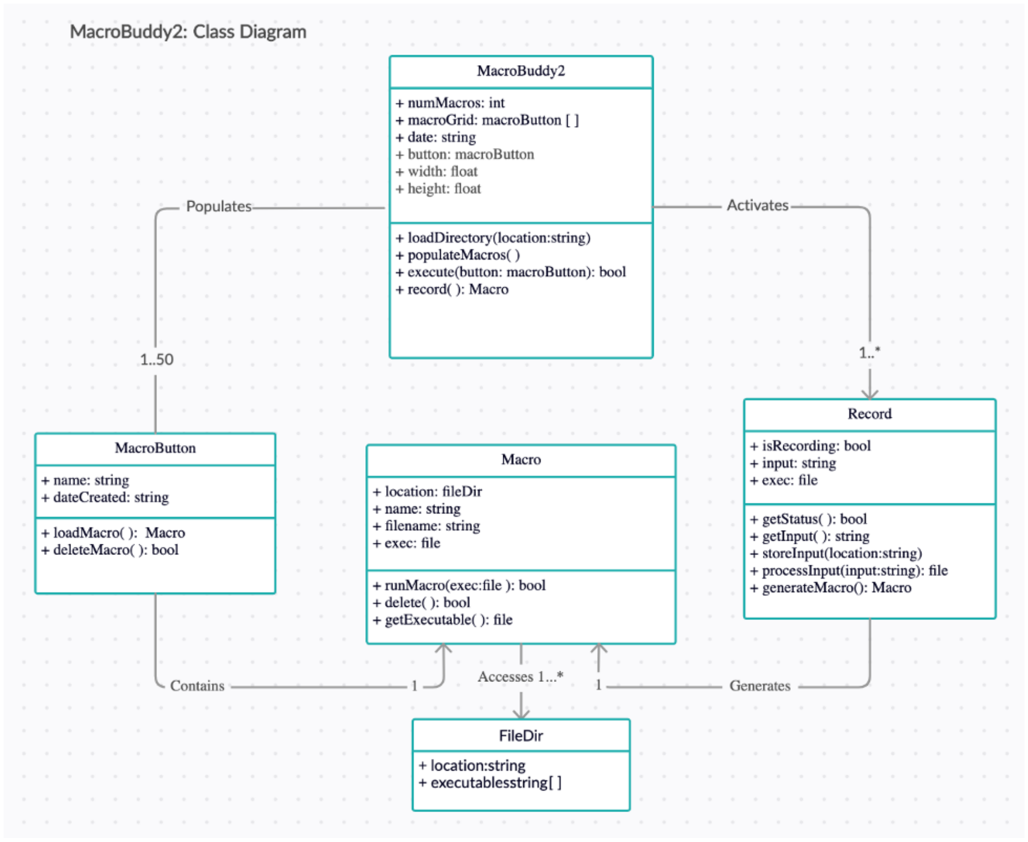
Use Case Name:	***Execute Macro***
Actors:	N/A
Description:	The Macro will execute on the terminal after loading.
Type:	Secondary
Includes:	Terminal
Extends:	N/A
Cross-refs:	
Uses cases:	Load Macro

Use Case Name:	***Terminal***
Actors:	User
Description:	The user will type commands in the terminal which are to be recorded. The user can run a macro on the terminal.
Type:	Secondary
Includes:	N/A
Extends:	N/A
Cross-refs:	
Uses cases:	Execute Macro

4.2 Class Diagram:

Due to the focus of MacroBuddy2 having one goal, which is to produce a macro from user input, our class diagram is split into the actions of the user interface. The UI manages the actions on screen, represented by the MacroBuddy2 Class. The Macro grid, which is seen in the MB2 class, contains a list of macro buttons, which contain the methods to manage that referenced macro, whose info is contained in the Macro class.

The other side of the diagram, in the record class, manages the user input. This class contains all the methods needed to process and generate the macros which will populate the macro grid.



Element Name	**Description**	
MacroBuddy2	The GUI window which contains a number of Macro Buttons and possesses buttons for running and recording macros as well as possessing variables pertaining to the drawing of itself as a window. This framework creates, pulls together, and populates macros.	
Attributes:	+ numMacros : int	Number of macros
	+macroGrid:macroButton[]	Vector of macro buttons. It contains all the macros the user created.
	+ date : string	Date formatted as an ascii text string is used to create logs of what operations occurred at what time
	+ button : MacroButton	This button represents the Macro instance. When the user clicks it, the macro which is connected to it will execute
	+ width : float	Width of the window
	+ height : float	Height of the window
Operations:	loadDirectory()	Fetches location of generated macros
	populateMacros()	When the user passes a path to a directory, this function will open the directory to find a file or save a file
	execute() : bool	Executes the macro that is connected to the button selected by the user
	record() : macro	Begins or ends user input recording

Element Name	**Description**	
MacroButton	A Macro Button is a user interfaceable means by which loading a Macro. Each Macro created and stores appears as a Button on the UI; clicking on this Button allows the user to load or delete the Macro. Macros appear with a name and they appear in order of date created.	
Attributes:	+ name : String	The macro’s filename
	+ dateCreated : String	The system time file which is used to populate the macros in the MacroGrid.
Operations:	+ loadMacro() : Macro	The function populates the button with the macro file
	+ deleteMacro() : bool	The function deletes a macro from the UI and user’s system.

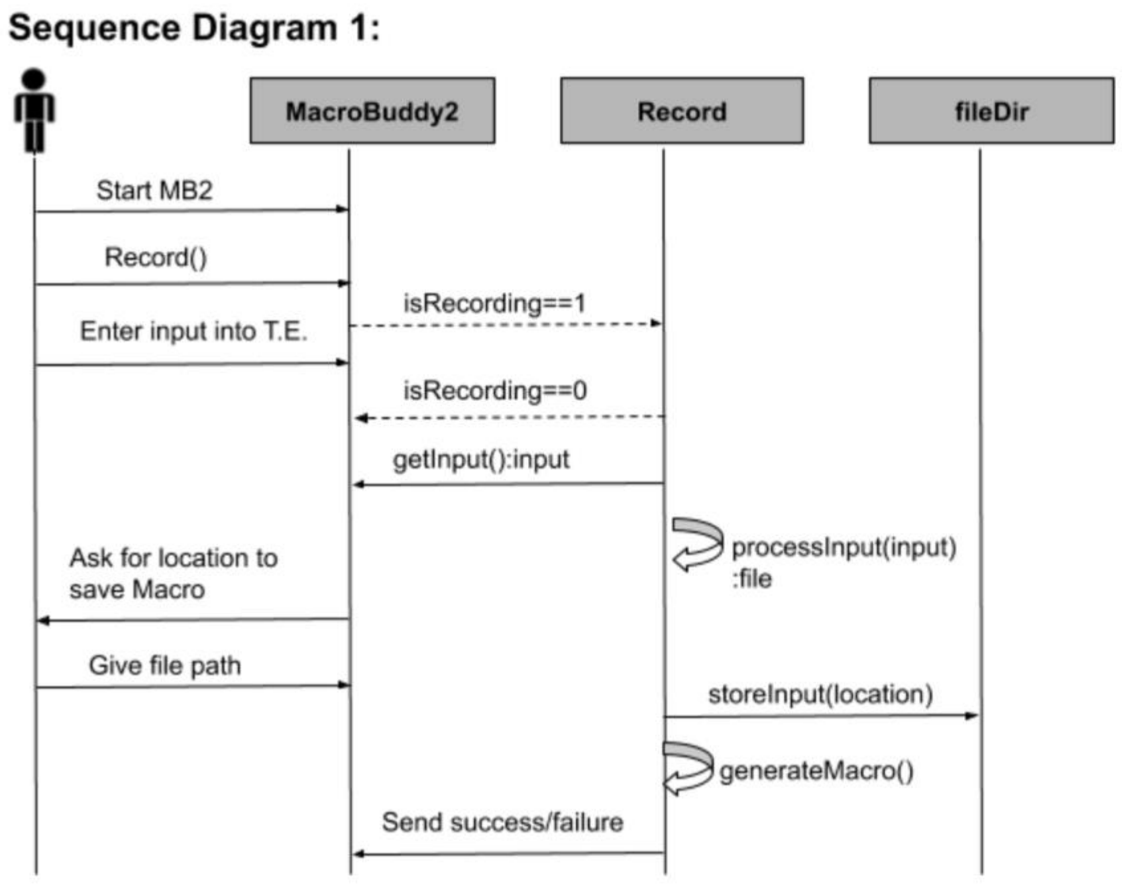
Element Name	**Description**	
Macro	The processed series of commands which the user has previously created. This element is a Macro object and thus contains all variables related to the definition and contents of a Macro.	
Attributes:	+ location : String	The Macro’s file path
	+ name : String	The Macro’s display name (name as appears in UI)
	+ filename : String	The Macro’s file name (name for the physical file)
	+ exec : String[]	Array of Strings containing file contents
Operations:	+ runMacro() : void	The function for running the Macro
	+ delete() : void	The function for deleting the Macro
	+ getExecutable()	The function for populating the Macro’s contents

Element Name	Description	
Record	The button which controls what the user's interaction with the terminal yields, if anything. Also controls the parsing and storage of the Macros.	
Attributes:	+ isRecording : bool	Variable for determining whether the emulator should scan user input
	+ input : String	Variable for storing the last line user entered
	+ exec: file	Variable for storing the executable file that contains the commands entered by user.
Operations:	+ getStatus() : bool	This function returns the TRUE if recording or FALSE if not recording
	+ getInput() : String	The function for saving the last line user entered
	+ storeInput(location:string)	The function for saving the Macro in a file path
	+ processInput(input:string) : file	The function for parsing the user input into Macro form. Then it will generate a file composed of these commands from the user.
	+ generateMacro() : Macro	This function will generate a macro object that will store the executable file name, its location, and the macro's name.

Element Name	Description	
fileDir	The fetcher for the Macros at the filepath directory	
Attributes:	+ location : string	Variable for the filepath
	+ executable : string[]	The variable for the list of Macros in the file path

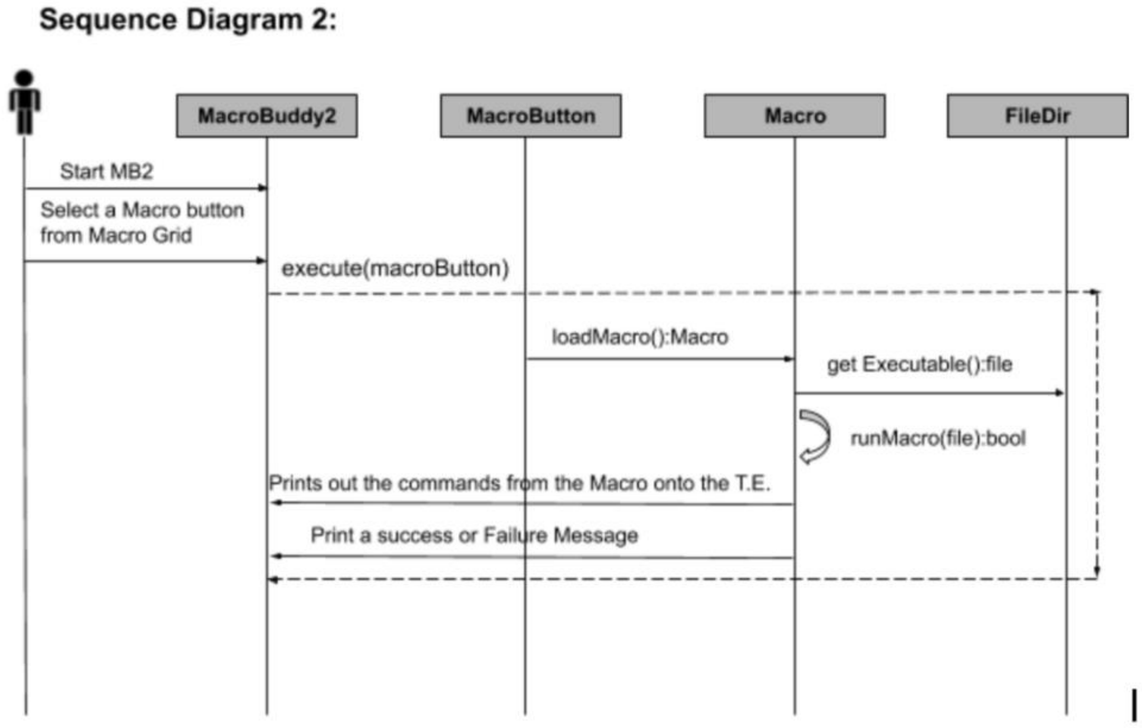
4.3 Sequence Diagram 1: User Records Input and Generates a Macro

This sequence diagram shows how the MB2 will record user input and generate a Macro. When the user selects the record button for the first time, MB2 will start recording the Terminal Emulator (T.E). When the user presses the record button again, the recording will stop. Between the start and stop of recording is where the user will enter the input(commands). While recording, MB2 will set isRecording status to True(1). Otherwise, it will set isRecording status to False(0). After the recording is stopped, MB2 will get the input using getInput() and the Record class will process the input and generate a file which contains the commands which can be accessed and executed later on. MB2 will now ask for a location where the macro can be saved, then saves it there. Then, a macro will be generated. This macro will contain the information about the file's location. Finally, the record will send a success/failure message.



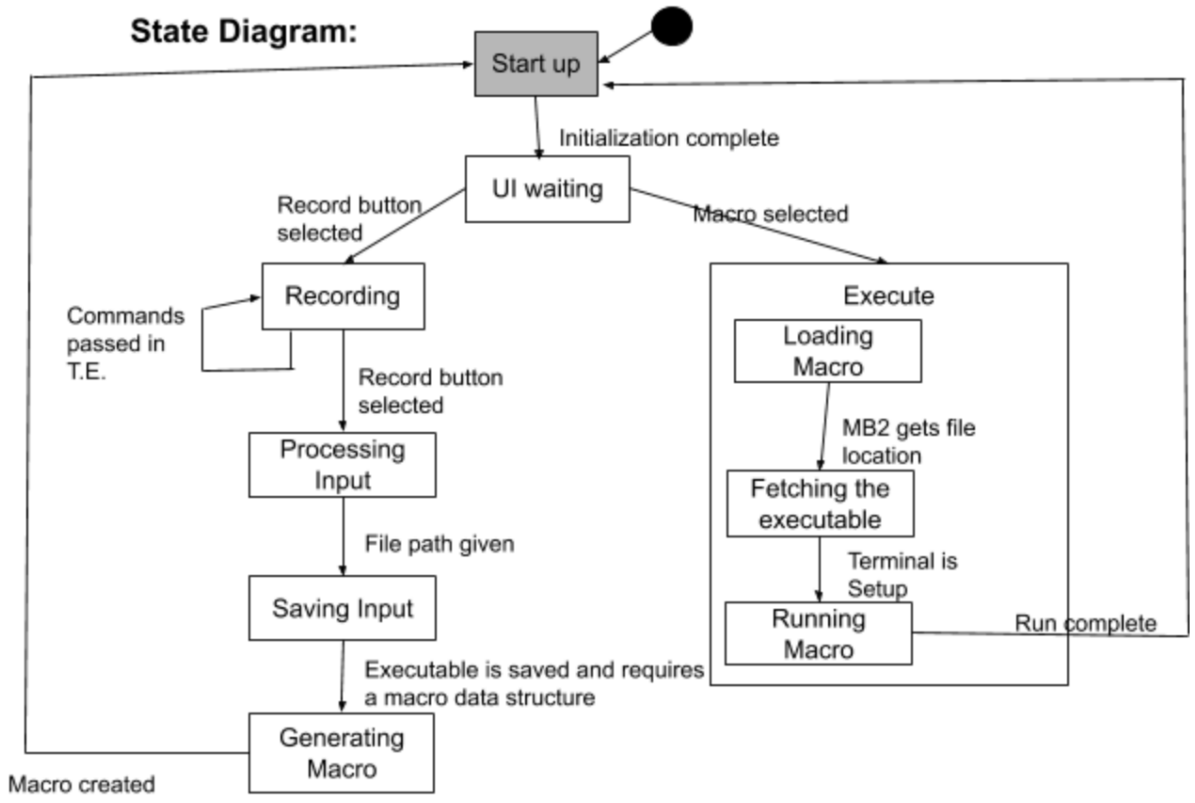
4.4 Sequence Diagram 2: User selects a Macro to Execute on the Terminal

The second sequence diagram shows the function of how MB2 loads a macro chosen by the user and executes it. When the user selects a Macro MB2 will run the execute(macroButton) function. In this function, macroButton will load a macro which will then fetch for the executable file in a location. Then it will run this file in the runMacro(file) function. This will print out all the commands onto the Terminal Emulator. After completion, there will be a success or failure message. This message will depend on return value of the execute(macroButton) function which is Success or Failure.



4.5 State Diagram:

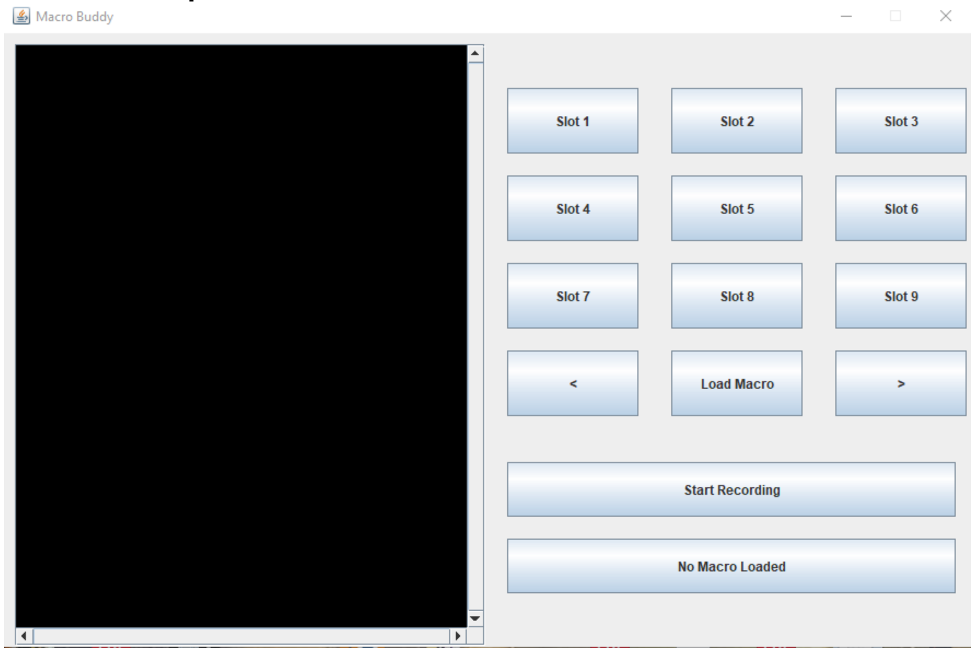
This diagram shows the expected states in the MB2 application and the order in which the MB2 application goes into these states. On the left, a recording sequence is illustrated; on the right, an execute sequence is illustrated. Many of these states are automated except for when capturing the user's input during the "recording" state.



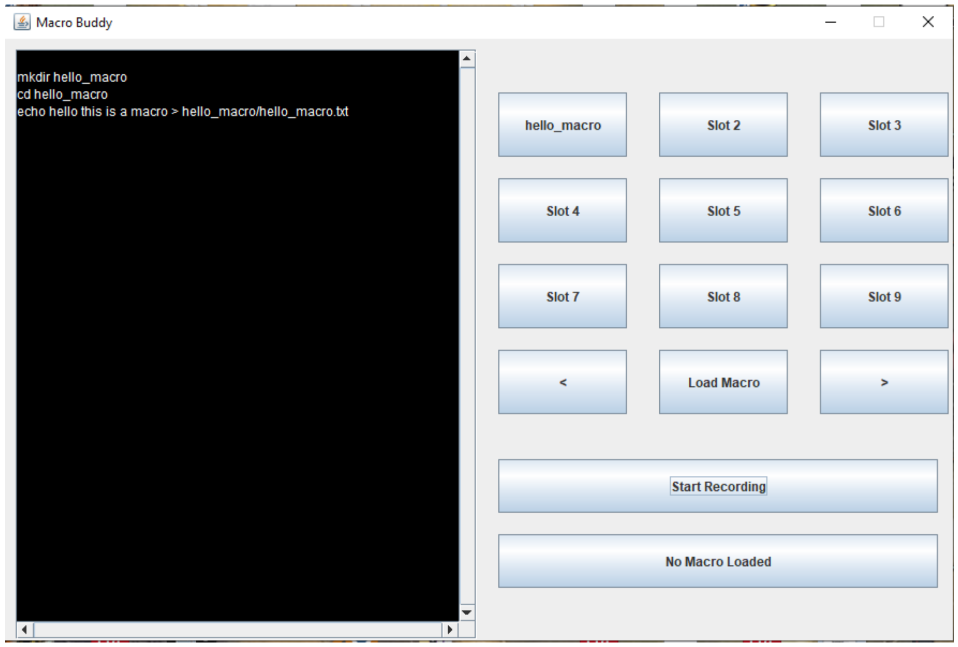
5 Prototype

UI – The UI of the prototype for Macro Buddy will fully implement all the necessary controls for a completely functioning product. In order to load already generated macros, the user will be able to scroll through pages of previously created macros. The user will be provided with an execute and record button for those functions. All button interactions are laid out on the right side of the UI, while the terminal window will be shown on the left half of the screen.

Initial Startup of MB2:

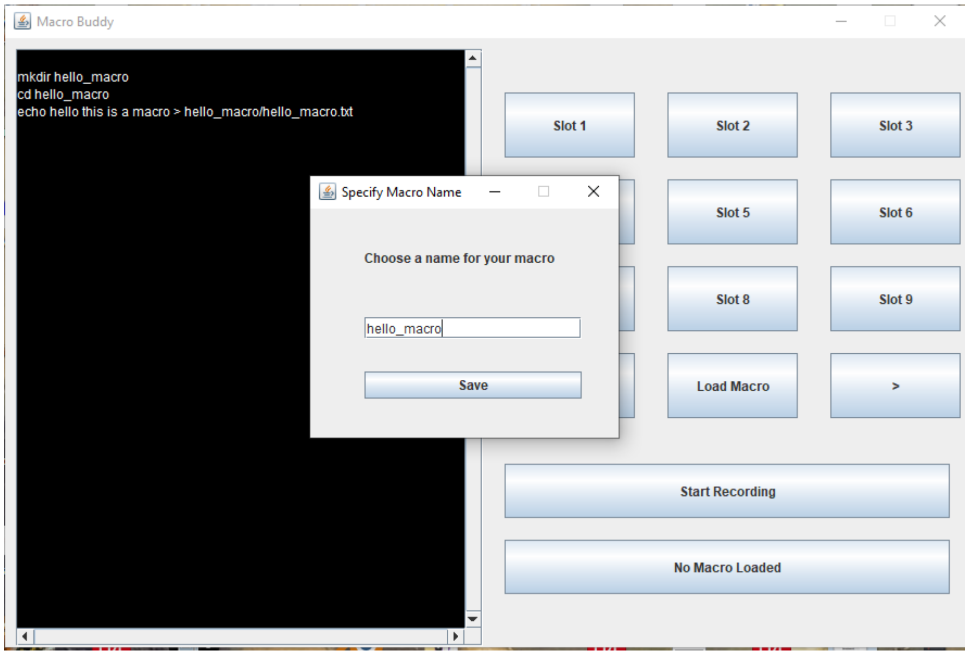


Recording Commands:



```
mkdir hello_macro
cd hello_macro
echo hello this is a macro > hello_macro/hello_macro.txt
```

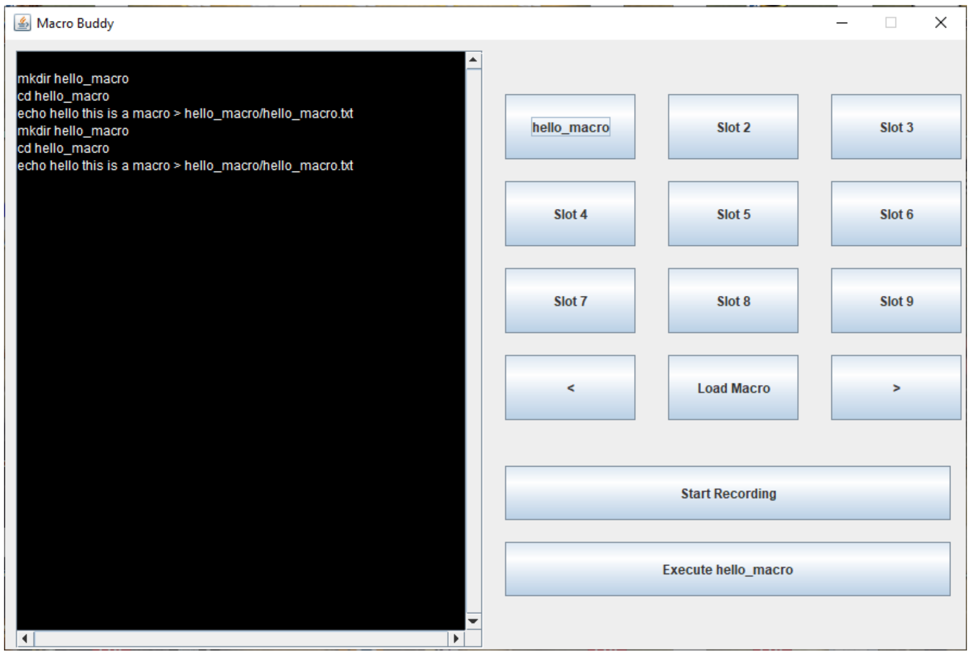
Recording Stopped and Saving Macro:



```
mkdir hello_macro
cd hello_macro
echo hello this is a macro > hello_macro/hello_macro.txt
```

hello_macro

Executing the “hello_macro”:



```
mkdir hello_macro
cd hello_macro
echo hello this is a macro > hello_macro/hello_macro.txt
mkdir hello_macro
cd hello_macro
echo hello this is a macro > hello_macro/hello_macro.txt
```

When you want to create a Macro using Macro Buddy, follow this process:

1) Boot up Macro Buddy
2) Press "Start Recording"
3) Type in the Macro Buddy terminal as if you were in command line
4) Press "Stop Recording" when done
5) Save using any name you want; the save file will be located in the directory macro buddy is located in /macros

When you want to load a Macro you created using Macro Buddy, follow this process:

1) Boot up Macro Buddy
2) Use the list of Macros that you have created to select a macro, use the arrows to navigate pages
3) Select a Macro
4) Execute the Macro

If you want to load a Macro stored in a non-default directory OR if your macro is not showing up in the button list, follow this process:

1) Boot up Macro Buddy
2) Use the Load Macro button
3) Execute the Macro

5.1 How to Run Prototype

Download Instructions: In order to download MacroBuddy, navigate to the website link below and choose a download based on your preferred OS. To run on MacOS, download the .jar file, for Windows download MBuddy2_v0.2.exe

Requirements: Java Version 15.0.1

Operating Systems: Works on Windows; tested on Windows 10, assumed to work on Windows 7/8/8.1. Works and Tested on MacOS Big Sur. Assumed to work on all recent versions of MacOSX.

Plugins: None

Constraints: None

Link: https://github.com/Allen3Ryan/MacroBuddy

5.2 Sample Scenarios

MB2 is best used when the user is using a set of terminal commands that are often repeated. For example, a frequent task of most PC users is to delete all files from the trash bin. Although a simple task through a command line, the user would have to change directories (cd) to the trash, and then invoke a delete all command from there. With MB2, the user can write the set of commands only once, and then in the future simply click on a button that references the set of commands and the system will automatically delete all files from the trash, thus saving time and potential missteps for the user.

Another potential scenario for use for MB2 is large scale testing. When tests are run on files in a static location, a macro could be written in order to easily test these files with a normally tedious set of command line prompts. Because the terminal can run any set of commands, MB2 would be able to automatically launch a set of tests on a file location specified by the user in the macro input terminal in the UI.

Another use case for this system is for source control setup for developers. Once again, because the system can run anything that a terminal can, it can be used with git as well. Often a developer will want to setup a new repository, but the process can be lengthy, and mistakes can be made. If a user creates a generated macro for this, all they must do is press a button in the macro grid and a new repository will be created within the project folder, which they can then move anywhere they like.

6 References

[1] https://jstar-c.github.io/MacroBuddy2/

Our Macro Buddy 2 website

[2] https://www.youtube.com/watch?v=5o3fMLPY7qY&t=651s

Java GUI Tutorial in Java Swing. Used to build UI skeleton.

[3] https://stackoverflow.com/

Reference for debugging and syntax correction

[4] https://www.w3schools.com/

Reference for HTML, CSS, and JavaScript used on the website

[5] https://getbootstrap.com/docs/3.4/

Reference for Bootstrap used on the website

[6] https://www.youtube.com/watch?v=5GcQtLDGXy8

Brad Traversy: Bootstrap

[7] https://www.youtube.com/watch?v=UB1O30fR-EE

Brad Traversy: HTML

[8] https://www.youtube.com/watch?v=yfoY53QXEnI

Brad Traversy: CSS

[9] https://github.com/Allen3Ryan/MacroBuddy

Prototype download

7 Point of Contact

For further information regarding this document and project, please contact **Prof. Daly** at University of Massachusetts Lowell (james_daly at uml.edu). All materials in this document have been sanitized for proprietary data. The students and the instructor gratefully acknowledge the participation of our industrial collaborators.