Extending Computer Interactivity Using Digital Musical Devices

Ethan Egerton

2067147

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science
of the
University of Portsmouth.

Project Supervisor: Alexander Gegov
School of Computing
Engineering Project | PJE40

2024

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

I confirm that I have read and understood the University Rules in respect of plagiarism and student misconduct.

I declare that this work is entirely my own. Each quotation or contribution cited from other work is fully referenced.

Date: 2024

Abstract

Musical Instrument Digital Interface (MIDI) controllers send signals to synthesisers to create sound. However, beyond this, their functionality is vastly limited. Giving more functionality to MIDI controllers can extend their use into other areas of work, such as programming, video editing, or general computer use. This report explores the development and design choices of creating an application which can perform specific tasks based on specific MIDI controller inputs. This project's key aims were to develop the desired software and implement usability theory to ensure a satisfying user experience.

Keywords: *MIDI, Digital Music, Application Development, Usability, Musical Hardware*

Acknowledgements

I thank my supervisor, Alexander Gegov, for his help and assistance throughout this project. Additionally, I thank my close friend, Hamish Chapman, who has always supported me emotionally throughout my academic studies.

Word Count

The word count in this document calculated by Overleaf is **10730**.

Consent to Share

I consent to this project being archived by the University Library and potentially used as an example project for future students.

Artefact Code

All the code developed in this artefact can be found at:
https://github.com/ethan-egerton/BSc-Dissertation-Artefact

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Background

Musical Instrument Digital Interface (MIDI) is a system that handles the connection and communication between electronic musical devices and computers. This connection allows both devices and computers to send musical and operational instructions to one another. Often confused with sound synthesis, MIDI is solely the communication of musical devices and computers.

A MIDI controller is a device that sends and receives MIDI signals. Often, a MIDI controller takes the form of a piano, but it can also look like a set of drums or even a guitar. Regardless of shape, all MIDI controllers serve the same purpose and function.

MIDI controllers can send signals to synthesisers, generating audio, often the pitch based on the signal from the device. MIDI controllers can also control other MIDI controllers, even multiple; this is often called daisy chaining.

MIDI instruments can be expensive; therefore, only using them for playing or creating music limits their potential as devices to assist workflow. If the functionality of a MIDI controller were to be expanded, it could be used as a customisable input device. As a customisable input device, buttons could activate hotkeys that can assist in programming, video editing, or general computer use.

MIDI controllers provide expressiveness in creating music. This expressiveness could be pushed into how users control computers instead of music. Unlike the binary inputs of a keyboard and mouse, knobs and other dimensions of music-based input could be used to control parameters of software or operating systems. Unfortunately, no software that meets this exact specification currently exists.

1.2 Aims and Objectives

This project aims to create an executable that can use the input of a MIDI controller to become more expressive and valuable than a keyboard and mouse. This software should be easy to access and simple to set up. The creation of this software should increase the usage of MIDI instruments and give MIDI instruments new purpose or functionality. A few objectives for this project are as follows:

1. Research MIDI instruments and usability concepts.
2. Identify existing solutions and features and find new features to improve functionality.
3. Develop easy-to-use software that can read MIDI signals to perform actions and control general parameters of the Windows 11 operating system.

These objectives are essential as they are quantifiable goals that can measure the success of this project; additionally, they are a set of steps to obtain the project's aim.

1.3 Constraints

Some constraints will need to be considered during the development of this project. The two main constraints are time and scope. There is only a limited amount of time to work on this project; therefore, special care will be given to ensure that deadlines are met. The scope will limit the features that are not necessary to make this project functional.

1.4 Considerations

Specific topics should be considered in the development of this project. To ensure its safety, the legal, ethical, professional, and social aspects should be discussed to outline its intentions clearly.

1.4.1 Legal

Data gathered or processed during the project will be handled appropriately following GDPRs. Participants in any data collection will be informed thoroughly about what data will be collected and how it will be processed. Explicit consent will be acquired before any information is gathered. Any personal data that might be collected will be minimised and anonymised to ensure that no participant can be identified through the data used in this project. Data stored during the project will be destroyed immediately after its development.

1.4.2 Ethical

In the previous subsection, ethical considerations were made about gathering user data. The project has no visible ethical issues as it focuses on helping computer users interact with their machines differently.

1.4.3 Professional

To maintain professionalism throughout this project, fundamental software engineering principles must be followed to ensure the software produced is high quality. These principles include maintainability, reliability, and efficiency. Maintainable software should be well-documented with well-written and structured code to ensure the project can be maintained and continued after development. Reliability revolves around ensuring the software does not contain bugs and is accessible at all times. Efficient software should use the most optimal resources the running machine offers.

To achieve the project's goal of creating software that allows MIDI controller users to interact with their computers in more creative ways, the software should be very accessible. After developing the software, it will be released as open-source.

1.4.4 Social

This project aims to allow MIDI controllers to be customisable computer peripherals. This could allow users with accessibility needs to map specific actions to MIDI controller inputs, bringing a larger audience to MIDI controllers and computers. On the contrary, MIDI controllers can be expensive and are not typical computer-related items. Therefore, the price of a MIDI controller may deter users from using this software.

1.5 Project Risks

At the project initiation stage, a list of potential risks that the project could encounter was explored and noted. Table 1.1 shows these risks, many demonstrating plausible risks and ways to mitigate or avoid them.

Description	Impact	Mitigation/Avoidance
Workstation failure or technical fault.	Loss of work or loss of time to produce work.	Backup artefacts to GitHub, written work to Google Drive, and work on university campus workstations in the case of workstation failure.
MIDI controller failure.	Unable to work on project artefact.	Have a small budget available to buy a cheap and elementary MIDI controller. Otherwise, ask the University if it would be possible to lend or use a MIDI controller for testing.
Loss of Internet.	Hinders access to the online copy of the written work or a live project version.	Try to make all work available offline. However, if not, work on campus.
Illness (short)	A few days of disruption.	Keep ahead of schedule and be prepared to continue work after disruption.
Illness (long)	Many days or weeks of disruption.	File for ECF (extenuating circumstance form) and contact the supervisor often.
Bereavements	Few days or weeks off.	File for an ECF (extenuating circumstance form) and contact the supervisor often. Attempt to be ahead of schedule to mitigate lost days.

Table 1.1: Project Risks

1.6 Project Plan and Development

This report will document the project development process, specifically why certain decisions were made, the justifications for the project's directions, and the artefact's implementation and testing. It will start with a literature review discussing MIDI and Usability. It will then explore the existing solutions to understand the basic requirements for a solution and how they can be improved.

During the project initiation stage, alongside the Project Initiation Document (see Appendix A), a Gantt chart, shown in Figure 1.1, was created to help plan the project's

stages, most notably the milestones to hit idealistically. Further into the project, the Gantt chart was changed to adapt to the issues that had risen throughout the project development.

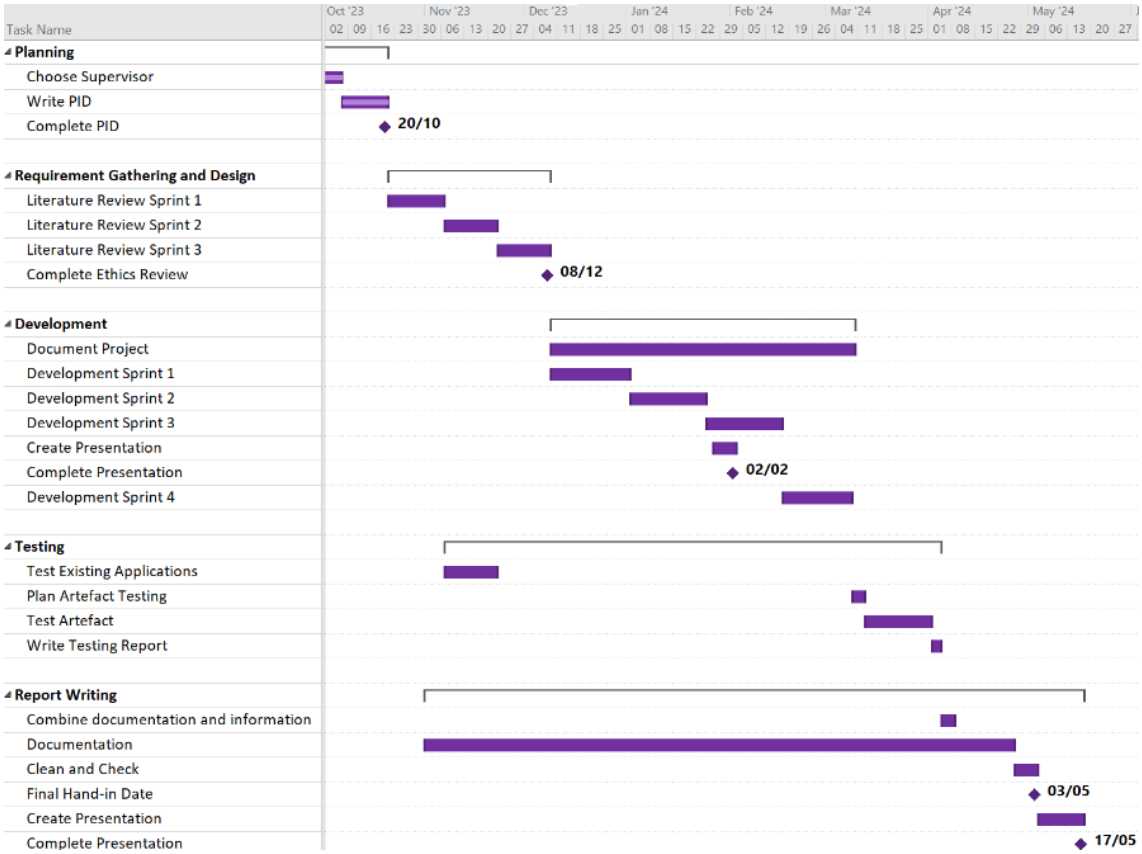


Figure 1.1: First Gannt Chart

1.7 Report Structure

The structure of this report starts with a literature review analysing the technology and techniques to be implemented into the artefact whilst identifying potential requirements that the artefact might need. After technological information is gathered about the topic, a light explanation about the software development methodology used in this project. In the Requirements chapter, requirements inspired by the literature review are outlined and prioritised. The Design chapter involves taking requirements and creating design plans and strategies to implement them. Given the design plan, the Implementation chapter explores the journey taken to develop the project. Once the artefact has been created, testing begins. The Testing chapter explores the different types of testing performed and their effectiveness. Following this are the Evaluation and Conclusion chapters, which analyse which requirements were fulfilled and how successful the project was.

Chapter 2

Literature Review

2.1 Introduction

This literature review focuses on usability and MIDI. Additionally, a section discusses artefacts similar to the project aims, what they provide, and requirements that could be improved. A review strategy was developed to ensure the information gathered is relevant.

2.2 Review Strategy

Researching these topics will reveal strategies for making software accessible and expose more technical knowledge about MIDI. Keywords such as "MIDI", "MIDI applications", and "Usability in Software" were used to find resources and relevant information about these topics.

As MIDI is not a highly academically covered subject, much information involving MIDI has been taken from blogs. Unfortunately, with the release of MIDI 2.0, the MIDI Association has removed MIDI specifications, which were available at the start of this project. Although internet blogs are not the most reputable sources of information, they provide decent insight into more technical aspects of MIDI.

2.3 Usability

There are five pillars of usability: Learnability, Efficiency, Memorability, Errors, and Satisfaction (Nielsen, 2024b). These pillars will be applied to the development of this project to produce a highly usable application.

Applying "Learnability" to this project means ensuring the artefact is simple and user actions are easy to identify and learn. Efficiency requires a user to be able to perform tasks, and to do so quickly, the artefact's User Interface (UI) must be quick to navigate so tasks can be performed more efficiently. Tasks must be easily remembered so proficiency in the application can be developed quickly. Errors must be prevented and easily rectified, and

the application must be pleasant and aesthetically pleasing. Implementing these factors will directly help implement the five pillars of usability.

However, more than just five critical lessons can be learnt from the study of usability.

2.3.1 Minimalist Design and Reducing Noise

To incorporate efficiency and satisfaction, the project should follow a minimalist design to help users focus on what is relevant in the application and for users to enjoy the visual look of the application (Fessenden, 2022). Refining what users see can create a sleek, clean aesthetic and enhance utility. However, decreasing the number of features and elements can rapidly hinder usability; thus, finding a good balance of simplicity and utility is essential.

2.3.2 Preventing Errors

Error prevention is critical to increasing usability as it deters users from making mistakes and generating frustration with the application. Error prevention can be implemented into application development by disabling actions if they are unneeded or inappropriate and only allowing the correct type of input (Shneiderman, 2016). For example, if a text box only allows a number input, refuse any input to enter the box that isn't a number. If errors do occur, ensure that any error is easily reversible.

2.4 MIDI

As previously mentioned, MIDI stands for Musical Instrument Device Interface, which is how specific types of devices share musical data. These specific types of devices are often called "MIDI controllers." A MIDI controller can send or receive MIDI data in all shapes and forms, such as a keyboard. MIDI controllers are regularly connected to a computer or synthesiser to generate audio (Mitchell, 2022).

2.4.1 MIDI Signals

MIDI signals hold musical data, such as musical notes, effect parameters, and the channel on which the data should be played. They are sent as messages the recipient can translate to perform a task, such as playing a specific sound (Gibson, 2016). These messages have five major components: the note integer, the activation integer, the channel number, the Control Change (CC) message, and the CC parameter value.

These messages can be sent via different methods, from the original 5-pin MIDI cable used to connect synthesisers and MIDI controllers in the 1990s to a more modern USB cable. There are two ways to connect MIDI controllers to a computer: the first is a direct USB, and the second is a MIDI interface. Most MIDI controllers have a USB port to connect to a computer; these ports come in various types, from USB A to Micro USB.

This is the direct USB method. The second method is used when a device does not have a USB MIDI port but a MIDI in and out. Gibson says, "MIDI (5-pin) cables are unidirectional", meaning to connect a MIDI device using 5-pin cables, a MIDI interface is also needed to transfer digital computer messages to the MIDI device and translate the analogue MIDI signals to digital for the computer. Either way, for a computer to use MIDI, it must be digital.

Expanding on the MIDI message components and understanding how the message is constructed will help manipulate it to the project's needs. The note number is between 0 and 127; this number is used to identify which musical note should be played. The activation number, also known as the velocity number, identifies how loud a note should be played; however, in the context of MIDI controllers, it is often used to describe how activated an input is (Vandenneucker, 2024). The channel number dictates which channel the note should be identified with. Different MIDI channels can be used to separate different MIDI signals. CC or Control Change values identify which parameter to change and what to change. CC values can represent effects like "Modulation" or "Volume" and reach up to 127 different effects. However, this will not be relevant to the project.

2.5 Previous Implementations

This section will explore the different existing artefacts and their strengths and flaws. A MIDI controller will be used to test the applications' functionality. The MIDI controller used is a Launch Control developed by Novation. This MIDI controller offers potentiometers, buttons and a MIDI channel changer. The MIDI channel changer will allow for testing whether an application can handle multiple MIDI channels simultaneously. Many more MIDI controllers exist and are available on the market; however, this device is cheap and offers most of the functionality of most MIDI controllers. This device does not offer MPE functionality; however, MPE does not have a significant presence in the current MIDI controller market.

These applications were found through Google search queries such as "MIDI to Keyboard", "MIDI to Macro", and "MIDI translator". Although these search results did provide promising findings, looking through internet forums to see which software large communities recommend yielded good results. The applications' functionality, usability, and ease of understanding will be tested. However, a deep understanding of MIDI and MIDI software has already been acquired through personal experience. Therefore, judging an application on how easy the software is to use for someone with no experience in MIDI software would assist in judging the application from a usability sense.

This section will help identify the requirements, features, and unneeded features for this

Figure 2.1: Novation Launch Control

project artefact. These applications will be judged reasonably harshly to avoid other developers' mistakes and ensure they perform exceptionally well. My criteria for evaluating these different applications were whether they were built for Windows and if they were free. If paid applications were evaluated for this research it would remove the point of filling the gap in the free MIDI translation application market. The two solutions meeting these criteria were "Midikey2Key" and "MIDI Translator Classic".

2.5.1 Midikey2Key

"Midikey2Key", developed by Rowoldt (2024), has an extensive range of well-performing features such as "MIDI Translation", "Application Execution", "Send MIDI", and some other miscellaneous features such as "Train Simulator". This application can also run in the background and on computer startup.

"MIDI Translation" is a feature that allows MIDI input signals set explicitly by the user to simulate keystrokes virtually. This allows the user to type out a sentence or activate a keyboard shortcut, such as "Control + C ", the copy text to clipboard command for the Windows Operating System. The "Application Execution" function allows the user to execute applications using the directory path to the application. "Send MIDI" sends specific MIDI signals back to the MIDI controller; this feature typically controls MIDI controller LEDs. Rowoldt also included functionality for combining a "Train Simulator" video game with MIDI controllers.

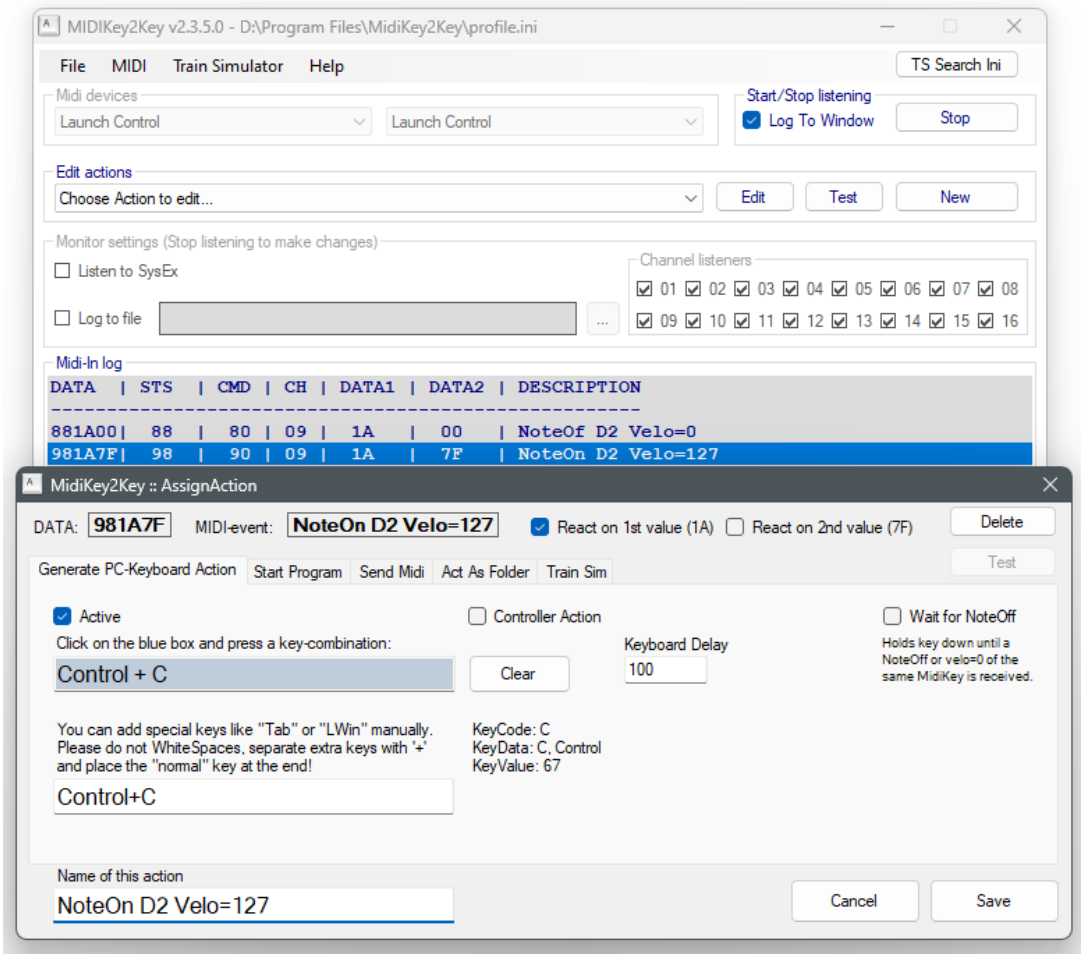


Figure 2.2: MIDIKey2Key Assigning Action to Specific MIDI Signal

To create an action, an action to be performed after receiving a specific MIDI signal from the selected device, the "New" button. This method allows for manual input of MIDI data, which is unhelpful to the average user as most MIDI users do not know the MIDI codes of their inputs. The other method, double-clicking a signal in the "MIDI Listening" section of the window, allows an action to be assigned to the signal; this is not obvious and was found on accident.

This application has many exciting features but lacks visual flair and general usability. It has many highly technical aspects, but its features feel hidden and obscured. Another major flaw of this software is that the user's list of actions is in a drop-down list labelled only by the MIDI signal it reacts to instead of a customisable name. This drop-down list was okay to navigate with small amounts of defined actions but would become overwhelming with many defined actions. Overall, the UI of this software could become overwhelming with popup windows and the many radio buttons.

To summarise the main features of this application:

1. MIDI to Keyboard Action

2. MIDI to Start Program
3. MIDI to MIDI
4. MIDI to Train Simulator
5. Visual MIDI signal aid

2.5.2 MIDI Translator Classic

"MIDI Translator Classic" by Bome Software (2019) is its deprecated version of "MIDI Translator Pro". This version provides the standard features of most MIDI translators but has a few extra features. Translation actions can be easily set using the "Capture MIDI" function, which reads MIDI signals from the selected MIDI controller and allows the user to select which signal and value the software should react to. Although more features would exist in the full-price version of Bome Software's

Additionally, this software allows for the naming of actions for customisability and ease of differentiating actions. The available reaction options are limited in this version; however, the "preset" feature is a handy usability feature which allows users to import and export application presets. This could build a community around the application and help users customise further.

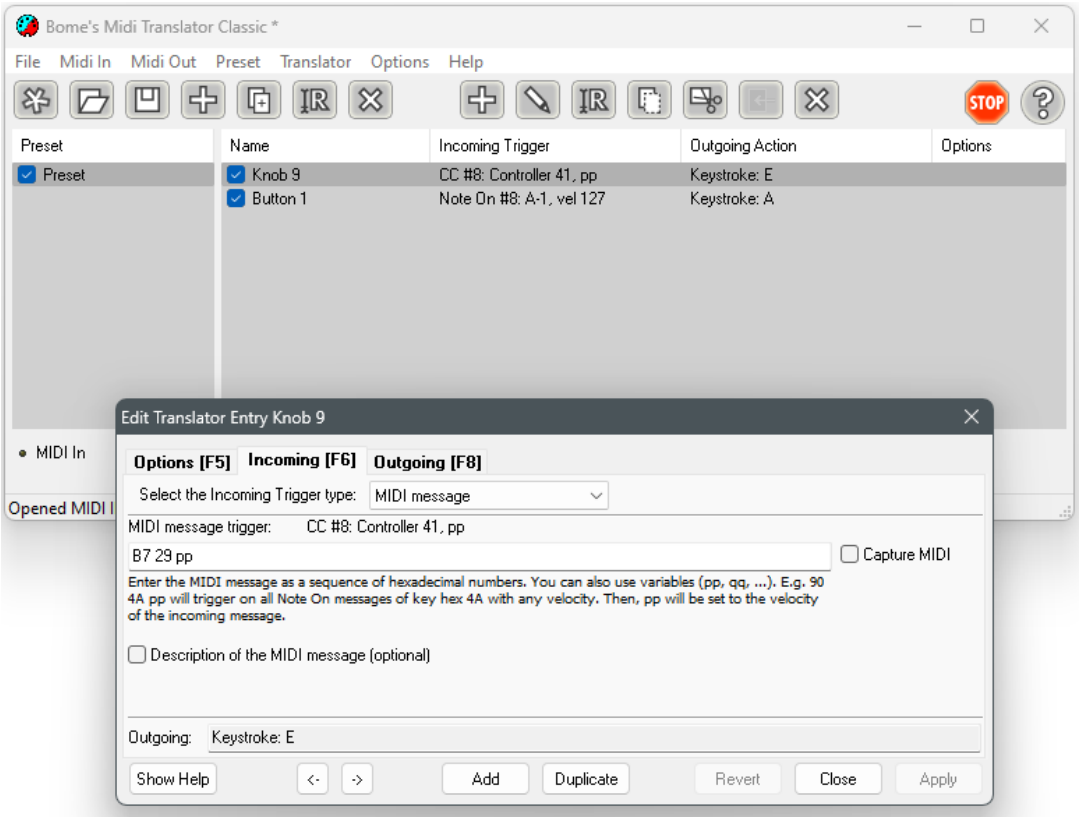


Figure 2.3: MIDI Translator Classic Assigning Action to Specific MIDI Signal

Little lights would activate at the bottom of the application when receiving or sending

MIDI signals. This was an excellent visual aid for the activity of the selected MIDI controller; however, it lacked any specific information about the signals being sent or received and thus felt counterintuitive.

To summarise the main features of this application:

1. MIDI to Keyboard Action
2. MIDI to MIDI
3. Record MIDI
4. Customisation of Translation (For example, presets and changeable names for translations)
5. Visual MIDI signal aid

2.5.3 Requirement Extraction

The main requirements become apparent based on the analysis of the existing solutions. The application must turn MIDI signals into keystrokes and MIDI. Customisation of the translation and visualisation of the MIDI signal should be in the application. The application could have in-depth customisation and the ability to record MIDI signals for translation, but it is unnecessary.

2.6 Conclusion

Interesting information was found about the different pillars of usability and some additional philosophies. MIDI and how MIDI signals function were explored, and the technical structure of a MIDI signal was revealed. Requirements about the existing solution were gathered. The five pillars of usability were Learnability, Efficiency, Memorability, Errors, and Satisfaction. Section 2.3, paragraph two, describes how to implement these pillars into an artefact.

Chapter 3

Methodology

3.1 Introduction

This chapter examines which software development methodologies exist and the reasoning behind the chosen methodology. Additionally, it discusses the planned project management techniques used in this project. The methodologies covered were Agile and DevOps whilst also discussing Kanban and Gantt charts.

3.2 Software Development Methodologies

Software is often developed following specific development methodologies which describe the process or steps by which it is created and maintained. There are many different software development methodologies; however, the more popular ones are more academically covered. Standard methodologies include Waterfall, Incremental, and Agile (Ruparelia, 2010).

3.2.1 Agile

Agile software development methodology starts with an initial planning phase, and a constant development cycle centred on adapting to feedback and identifying new requirements. Alsaqqa et al. (2020) suggested that the "goal for the agile methods is to reduce the overhead in the software development process with the ability to adopt the changes without risking the process or without excessive rework".

Agile has four core values and twelve principles; however, the four core values are essential to agile. While agile's general shape and flow are important, the core values separate agile from other cyclical software development methodologies. The core values of agile are as follows:

1. Individuals and interactions over process and tools.
2. Working software over comprehensive documentation.

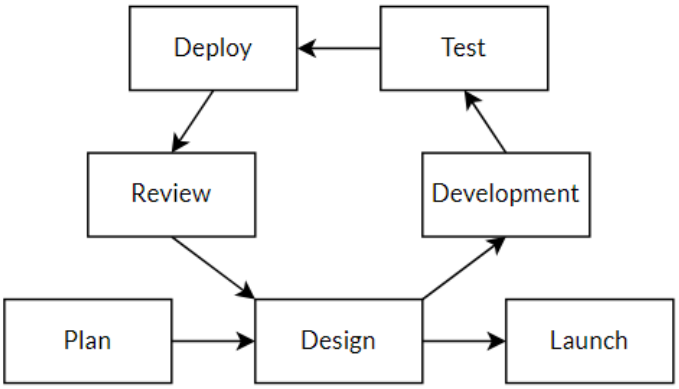


Figure 3.1: Agile Software Development Model

3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

These values should be applied to the development priorities of how the user uses the software instead of following a strict ideology of how the software should be built.

3.2.2 DevOps

DevOps combines "Development" and "Operations", and the methodology follows suit. DevOps is a collaboration between software developers and the IT operations team to increase the efficiency of "development to deployment and infrastructure monitoring" Ebert et al. (2016). There are two main phases of DevOps: Build and Deployment.

DevOps is similar to incremental development methodologies; however, like agile, it follows a set of practices to maximise efficiency and project quality. AWS (2024) states that the DevOps practices are "Continuous Integration", "Continuous Delivery", "Microservices", and "Communication and Collaboration". DevOps relies on constant production and deployment to gather constant feedback to power the development loop. This methodology is usually used for large-scale projects with long-term ongoing support.

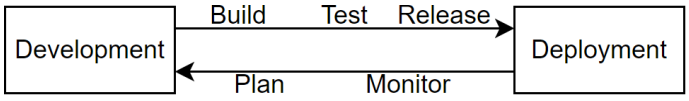


Figure 3.2: DevOps Software Development Model

3.2.3 Kanban

Kanban, a Japanese word meaning "Signboard", is a productivity framework where work items are visually represented in sections respective to the item's completion state. This method drives efficiency in teams by "orchestrating seamless task progression" (Radigan, 2024). This framework works well with Agile or DevOps in their feedback-to-development cycles. The standard stages of Kanban work item states are "To Do", "In Progress", and "Completed", but a Kanban board can be altered to fit any state.

Kanban can also be used well in team collaboration and solo development. Team members can be assigned individual work items, and solo developers can use Kanban to assess the project's state and rank tasks in order of importance.

3.2.4 Chosen Method

The methodology to be used for this project will be Agile. Agile provides an excellent sprint-based flow, which will be helpful as more is learned about the project and its needs as time progresses. Kanban will also be used to keep track of work items and the project's progress. Visually representing the progress will also help prioritise which requirements are more critical than others. This will be especially useful closer to the deadline.

Github, a service that helps with code version control, provides a "Project" view for a code repository. This project view allows users to create a Kanban board for their code repository, thus enabling good integration with code and project management. This will be used to keep track of the project in one central repository.

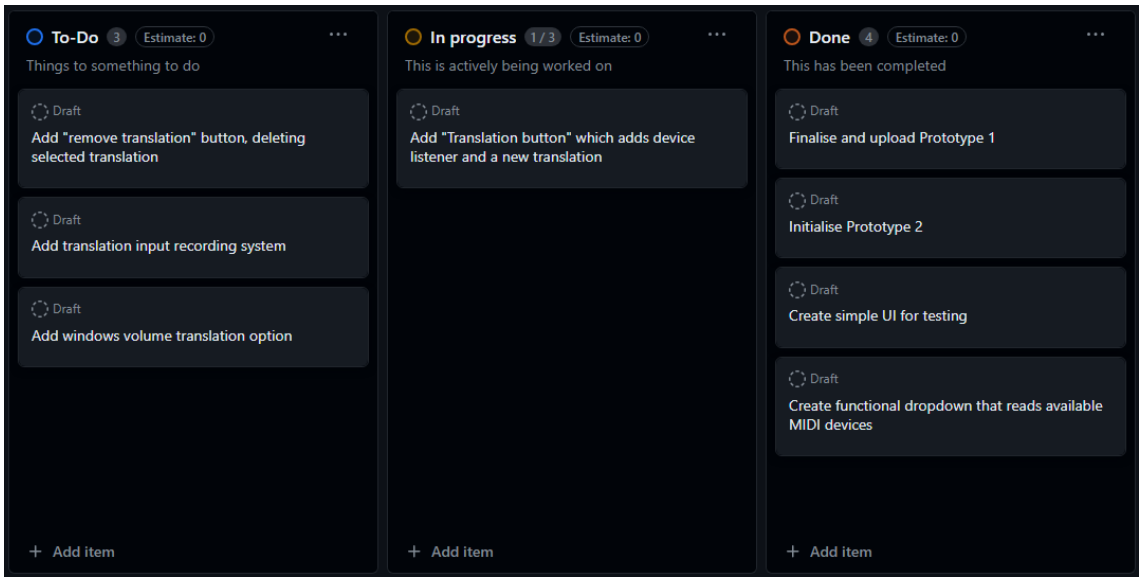


Figure 3.3: GitHub Kanban Board

Knowledge about the specific tools used for this project is unknown. Therefore, it would be ideal to have a methodology that revolves around not truly knowing the project entirely, such as Agile.

DevOps is certainly a helpful methodology. However, it revolves around building and deploying. I will not deploy this application during development; therefore, it is unsuitable for the overall setting. DevOps also often requires collaboration between people knowledgeable in development and people knowledgeable in deployment. This project is a solo development, and thus, it would be too large of an unnecessary workload for this project.

3.2.5 Implemented Methodology

The agile methodology has been adapted to suit the solo development of this application. In agile, the end of an iteration involves some review or feedback to alter the project's course, which seems complicated without much outside assistance with programming. To work around this, the feedback section has been removed, and the project will be developed in five stages: mock console application, simple GUI application, refined GUI application, testing, and evaluation.

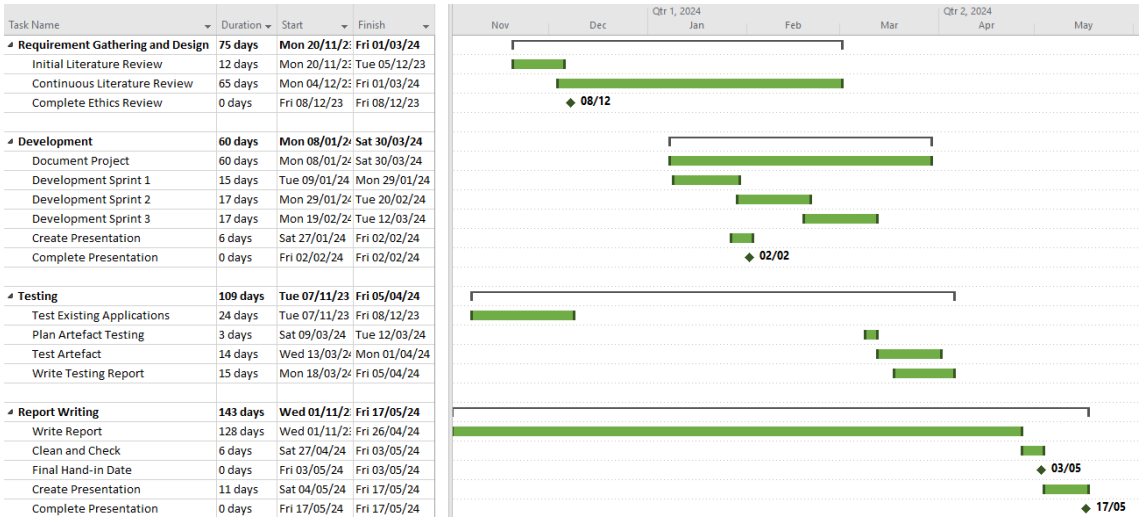
The mock console application stage aims to test the frameworks' functionality to become more accustomed to them. The simple GUI application stage aims to put functionality into a GUI to understand better how both UI and code can be combined. The "Refined GUI Application" stage aims to clean the code and UI while adding additional usability features. The testing stage will be a simple think-aloud user test to identify any weak points in the application's usability.

3.3 Project Management

During the project initiation stage, a Gantt chart was created to make a general plan for the project. This Gantt chart is discussed and shown in Figure 1.1. The project did not go according to that Gantt chart, and there was a lot of slippage and misunderstandings regarding the project scope. An additional Gantt chart was created to rectify and maintain the project structure. The updated version of the Gantt chart can be found in Figure 3.5. The key differences between the deprecated Gantt chart and the new Gantt chart are the literature review and the development. Initially, the literature review was planned to be done in three sprints. However, it quickly became apparent that this was unrealistic and that this project's literature review would grow as the development proceeded. This project had significant time slippages, delaying its start in November and December due to other university work. The development would realistically be a lot quicker than initially planned.

3.4 Conclusion

The software development methodology chosen was altered Agile with a Kanban and Gantt chart to keep control over the project. After considering Agile and DevOps, Agile was a better choice for solo development. The Gantt chart was renewed and updated to match more realistic deadlines and milestones.



Task Name	Duration	Start	Finish
Requirement Gathering and Design	**75 days**	**Mon 20/11/2**	**Fri 01/03/24**
Initial Literature Review	12 days	Mon 20/11/2	Tue 05/12/23
Continuous Literature Review	65 days	Mon 04/12/2	Fri 01/03/24
Complete Ethics Review	0 days	Fri 08/12/23	Fri 08/12/23
Development	**60 days**	**Mon 08/01/2**	**Sat 30/03/24**
Document Project	60 days	Mon 08/01/2	Sat 30/03/24
Development Sprint 1	15 days	Tue 09/01/24	Mon 29/01/24
Development Sprint 2	17 days	Mon 29/01/2	Tue 20/02/24
Development Sprint 3	17 days	Mon 19/02/2	Tue 12/03/24
Create Presentation	6 days	Sat 27/01/24	Fri 02/02/24
Complete Presentation	0 days	Fri 02/02/24	Fri 02/02/24
Testing	**109 days**	**Tue 07/11/23**	**Fri 05/04/24**
Test Existing Applications	24 days	Tue 07/11/23	Fri 08/12/23
Plan Artefact Testing	3 days	Sat 09/03/24	Tue 12/03/24
Test Artefact	14 days	Wed 13/03/2	Mon 01/04/24
Write Testing Report	15 days	Mon 18/03/2	Fri 05/04/24
Report Writing	**143 days**	**Wed 01/11/2**	**Fri 17/05/24**
Write Report	128 days	Wed 01/11/2	Fri 26/04/24
Clean and Check	6 days	Sat 27/04/24	Fri 03/05/24
Final Hand-in Date	0 days	Fri 03/05/24	Fri 03/05/24
Create Presentation	11 days	Sat 04/05/24	Fri 17/05/24
Complete Presentation	0 days	Fri 17/05/24	Fri 17/05/24

Figure 3.4: Updated Project Gantt Chart

Chapter 4

Requirements

4.1 Introduction

This chapter outlines how requirements for the project artefact are formatted and the different requirements extracted from the literature review separated by functional and non-functional. A brief discussion about MoSCoW and how requirements are classified.

4.2 Requirement Classification

Classification of requirements will allow the focused development of features to achieve the project's goals and the development of lesser-needed features if possible. The classification method will be the MoSCoW developed by Clegg and Barker (1994). The MoSCoW method is famous in agile and Rapid Application Development (RAD) development, and although pronounced as "Moss-cow", the O's are purely for pronunciation. The acronym stands for "Must, Should, Could, Won't" and defines the four main categories of MoSCoW as "must have", "should have", "could have", and "will not have", referring to the importance of requirements in the project.

Features will also be split into two main categories: functional and non-functional. Functional describes the artefact's functions and methods, while non-functional describes its general properties.

Definitions for the MoSCoW classifications with this project are as follows:

1. "Must have": Features that are imperative to this project to achieve its goals and targets
2. "Should have": Features that significantly improve the overall quality of the project but do not necessarily contribute to its core purpose should be added if resources allow it.
3. "Could have": Features that might improve the quality of the project but do not relate to the project's core goals and, therefore, could be added to the project if

resources are vastly available.

4. “Will not have”: Features not to be included in the project, whether to make room for more resources to be given to more necessary features or to make clear that it does not suit the project at all and has no place in the project.

4.3 Functional Requirements

ID	Requirement	Description	MoSCoW
01	Detect all signals from a MIDI controller	To read signals from a MIDI controller, allow the user to assign the signal to an action and then fire said action every time that signal is detected.	Must
02	React to specific MIDI signals	Allow a user to set specific actions that would fire every time a specific MIDI signal is detected from a device.	Must
03	Select MIDI controller	If two or more MIDI controllers are connected, the option to only listen to the signals from a specific MIDI controller should appear.	Must
04	Control Windows OS volume	To prove basic integration functionality with the Windows Operating system. As per the aims of this project.	Must
05	Simulate keyboard input	To prove shortcut macro functionality. As per the aims of this project.	Must
06	Choose MIDI Signal	Allow for input of the specific MIDI signal number to be detected for a reaction.	Must
07	Display MIDI Signal	Display the current MIDI signal number so it can be inputted into the MIDI signal selector.	Should
08	MIDI Signal Recorder	Record inbound MIDI signals for a selected translation and make the recorded MIDI signal the selected signal.	Should
09	MIDI to MIDI	Send MIDI signals back to the selected MIDI device	Should
10	Execute Script	Execute an executable application when reacting to a translation.	Could
11	Run in the background	Be able to run and execute translations whilst the application is closed.	Could
12	Velocity Specific Translation Activation	Translations activate specific MIDI signal velocities.	Could

Table 4.1: Functional Requirements

4.4 Non-Functional Requirements

ID	Requirement	Description	MoSCoW
01	User-Friendly Design	Easy to learn UI and system.	Must
02	Aesthetically Pleasing Design	Beautiful, interesting and modern design aesthetic.	Must
03	Open Source Software	Code released under an open source license.	Should
04	Help users understand the application	Create a guide document or in-application help function.	Should

Table 4.2: Non-Functional Requirements

4.5 Conclusion

All 15 requirements are weighted by importance using the MoSCoW method. The requirements were split into two main categories: function and non-functional. Functional requirements concern the technical aspects of the application, while non-functional requirements concern the application's design.

Chapter 5

Design

5.1 Introduction

This chapter discusses the actions and decisions made throughout this project's development process, from the programming language to the UI design. Covering which type of application is best for the application, how the users will interact with the software, and which open-source license is best for the projects goal.

5.2 Application Type

The three main types of applications to choose from are based on the type of device the application is run on: Mobile, Web Application, and Desktop. The primary goal of this project is to create an application that performs operating system actions based on MIDI inputs; the choice of type of application is obvious.

Web applications cannot perform many operating system actions besides copying text to the system clipboard. Although web applications can access and read MIDI controller inputs using the Web MIDI API developed by Mozilla Developer Network (2024), they lack functionality for controlling the operating systems.

With a few exceptions, mobile applications also lack control over the operating system they are executed on. The most mobile applications can do is control device volume. Additionally, using an external input device on a mobile device is often unneeded.

Desktop applications allow the most control over an operating system, from easily simulating keyboard operations to operating system actions like controlling volume.

5.3 Programming languages

Different programming languages and frameworks were considered for this project, and previous personal experience in Python, C#, and JavaScript encompasses proficiency in those languages, which are good starting points for this decision. Starting this project in

a language without previous experience could jeopardise this project as the inability to create the software effectively could severely hinder time and resources. A language that would more accurately fit the needs of this project needs to be selected.

Extended experience with Python has granted me extreme proficiency in the language, though some vital flaws are associated with Python. Python has an extensive catalogue of code libraries ready to use out of the box, and a few libraries assist in connecting MIDI controllers, which previous projects have provided great experience. The issue with Python is that developing a Graphical User Interface (GUI) can be difficult and frustrating. Despite its extensive code base, Python lacks GUI libraries that are both powerful and high-quality. This project will be developed using a language with robust GUI tools to achieve usability goals.

JavaScript combined with NodeJS, a framework that allows desktop application development through JavaScript, has many packages for MIDI and keyboard simulation, volume control, and others. An open-source package called “NodeGUI” is also used to develop graphical user interfaces with CSS and JavaScript. Other GUI packages are available, such as React Native, a User Interface (UI) framework created by Meta. However, although existing experience with JavaScript is available, there is no experience with either MIDI or GUI packages.

C# is an object-oriented programming language developed by Microsoft. .NET is Microsoft’s framework for C#, which is used for making desktop, web, and mobile applications. C# has a large community of developers who release packages for the public to use. DryWetMIDI is a package made by Melanchall (2024) that can access MIDI controllers and read their signals. .NET has a GUI library within it that is powerful and easy to use.

Python’s GUI capabilities are lacking and will not be used in the project. JavaScript and C# are both commendable choices. However, I have more experience in C# and .NET. In the case of failure to use C#, JavaScript will be the next best choice.

5.4 User Interaction

To fully understand how a user will interact with the software, a use case diagram was created using the web tool Draw.io (2024) to represent the connection between the user and the software. The diagram is shown in Figure 5.1 and only represents the fundamental interactions between user, software, and system. The user has three significant interactions with the software: "Add Translation," "Change Translation," and "Remove Translation."

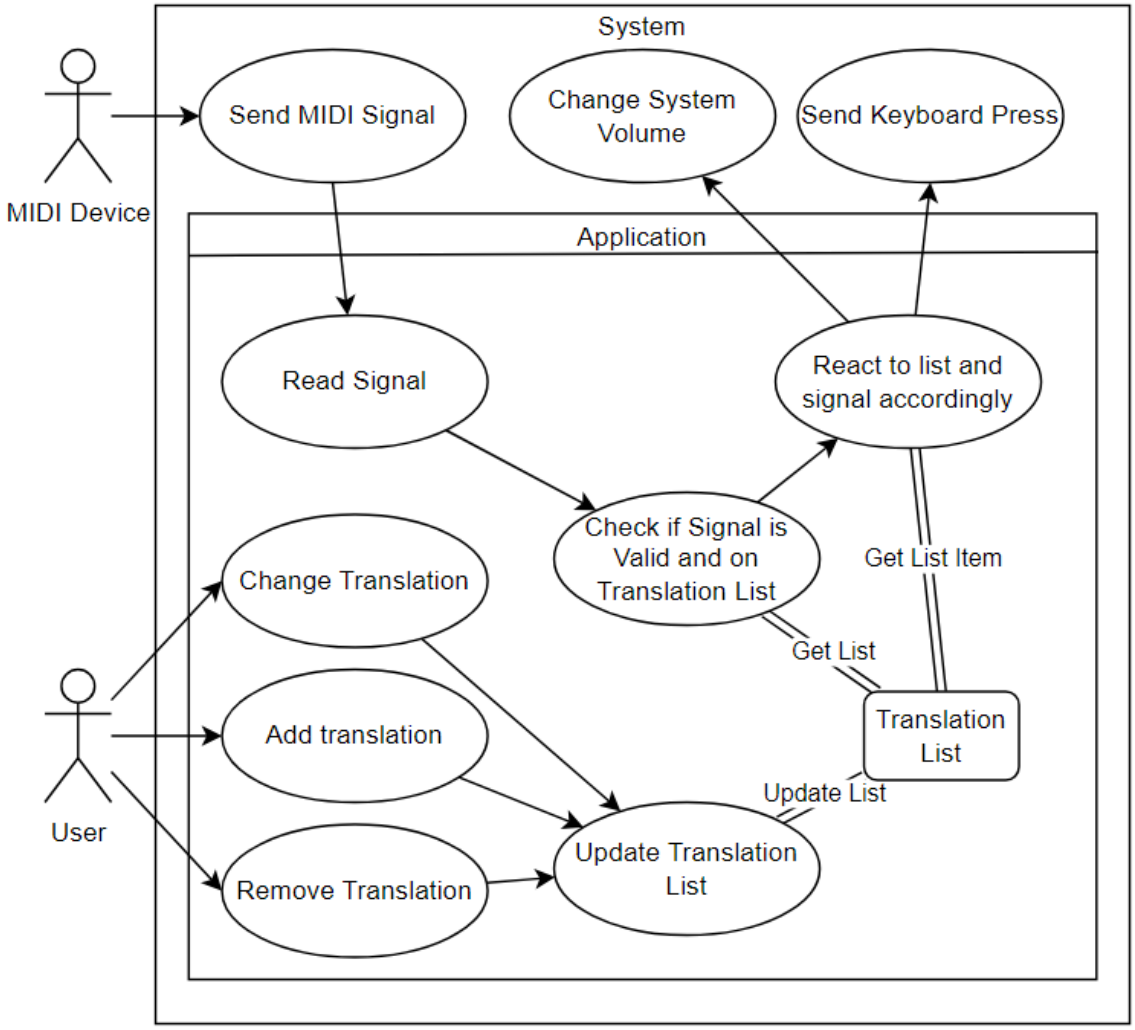


Figure 5.1: Use Case Diagram

5.5 UI Design

When planning the UI for this application, multiple designs were created to find the most suitable design for the artefact. This method was used to create a clear and revised plan for the development stage of this project. Multiple designs were made to narrow the design choices and weed out bad ones.

Many of the designs were inspired by already existing artefacts in the market. This inspiration was taken because these UIs fit the applications' use. The final design can be found in Figure 5.2, and the other discarded designs are in Appendix C.

After a series of different low-fidelity prototypes, one was picked out, refined further, and developed into the design in Figure 5.2. A few major design choices separated it from other designs. This design is very simple and has more room to expand outwards. With little to no visual clutter, the concise amount of inputs should allow for faster learnability. Ideally, having the "MIDI Number" in a highly visual area would give users more

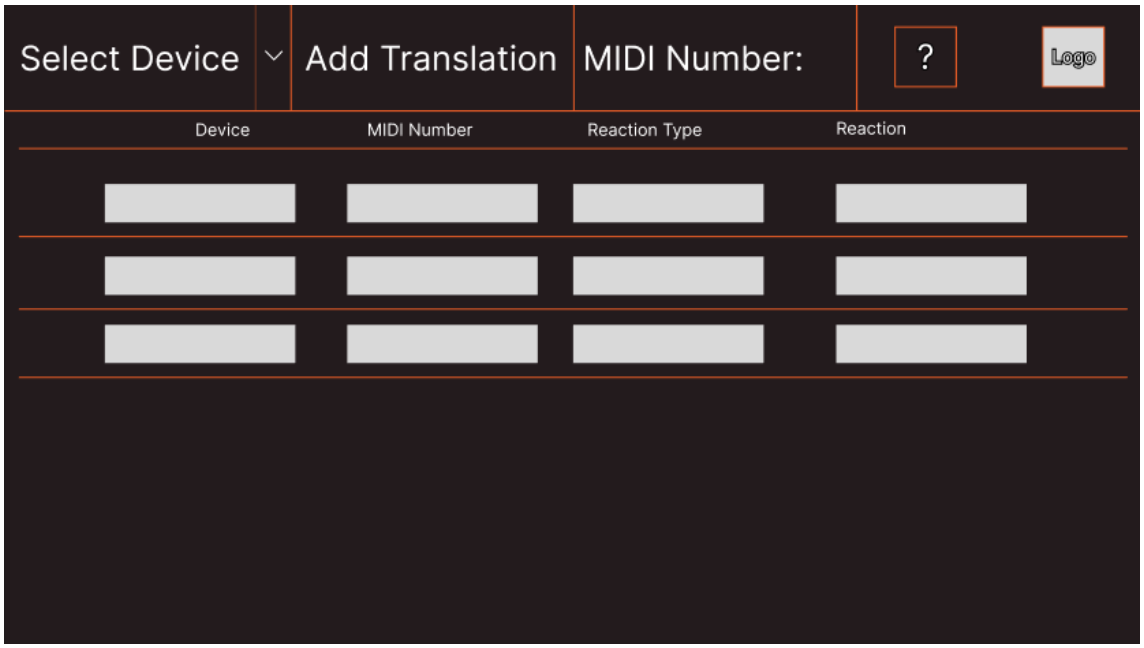


Figure 5.2: Final UI design

information about their device and which MIDI signal number corresponds to which number.

5.6 Open-Source Code

To make this software as accessible as possible, the project aims to release it as open-source. Open-source software is software whose code is publicly available for anyone to use under a specific licence. Many licences cover precise rules; however, the main two are the MIT license and the GNU GPLv3 licence. The MIT license allows any user to do almost anything with the code. However, the GNU GPLv3 allows users to do almost anything with the code except distribute closed-source versions. To ensure the code written in this project is not reused to create a closed-source, less accessible version of this project, this project will be released under the GNU GPLv3 licence.

The code developed in this project will be designed to make it easy for other developers to pick up and understand the system. Applying simple clean code methods will allow this open-source code to be easily maintainable. Clear variable names, good code comments, refactoring and modularisation of code, writing tests, and documentation will all be applied to achieve this goal.

5.7 Conclusion

The application to be developed will be a Desktop application made using C# and .NET. The final design was created using iterative lo-fi UI designs. Additionally, all code developed in this project will be released under the GNU GPLv3 open-source license. A

use case diagram was made to illustrate the relation between the user and the application.

Chapter 6

Implementation

6.1 Introduction

This chapter is split into three main sections, one for each development sprint for the project application. Additionally, one final section discusses the user guide developed for the application.

6.2 Learning New Technologies (Iteration One)

The first iteration aimed to build a simple console application with straightforward functionality to translate MIDI signals from a specific MIDI controller to keyboard input. This goal will help me understand the different technologies used in sprint two.

6.2.1 C# and Visual Studio 2022

To build C# applications, Visual Studio, an Integrated Development Environment (IDE) developed by Microsoft, is used to develop, debug, test and compile C# applications. Visual Studio 2022 is the most recent version of Visual Studio. While older versions are available, using the most supported, most up-to-date version made more sense to get access to the most recent tools used for developing in C#. Visual Studio 2022 comes in three versions: Community, Professional and Enterprise. Community is free of charge and includes the essential development tools; Professional is similar but provides tools for small teams and access to web services. Enterprise gives access to all Visual Studio has to offer, from extensive testing tools to cross-platform services. C# applications can still be written on the Community version, but features like web service tools are unnecessary for this project. To save resources, the Community version will be used to develop this application.

For this iteration, the "Console App" template was used to generate the ".csproj" file, a project file containing compilation information about the application and other necessary build files. The project file helps specify the information needed about the project itself, and to tell the compiler to use the .NET 8 framework.

6.2.2 Packages

Three main packages are used in the first sprint: Windows Forms by Microsoft, DryWetMIDI by “Melanchall”, and RtMidi.Core by “Micdah”. These packages were found by searching the Nuget website using the search term “MIDI”. Nuget is the package manager for .NET. Nuget is also built into Visual Studio and allows easy installation of packages. Two main MIDI handler packages were found, although DryWetMIDI has substantially more downloads and support than RtMidi.Core. With my previous experience with RtMidi in other programming languages, I chose to test both to see which was more effective.

Windows Forms is a predecessor to Microsoft’s Windows Presentation Foundation (WPF), primarily used to develop GUI applications. Although it was initially released in 2002, Windows Forms is included in most .NET releases. Although Windows Forms is designed for GUI applications, it contains classes and functions to manipulate Windows, specifically the ability to send key presses to Windows.

RtMidi.Core is a C# .NET wrapper for RtMidi, a C++ library that connects MIDI controllers to Linux and Microsoft and supports .NET 8. RtMidi’s functionality allows for the receiving and sending MIDI signals after initially using RtMidi.Core proved challenging to set up device listeners for MIDI controllers, and receiving MIDI signal events was also problematic as it required much code to update signal events.

DryWetMIDI is a .NET library that works with MIDI data and devices. It is available for Windows and macOS and supports .NET 8. Despite having no experience with DryWetMIDI, the design and functions were similar to my previous experiences with RtMidi in other languages and had more manageable code. An issue faced with DryWetMIDI was the device listener occasionally and randomly stopped working, which would stop the whole console application.

6.2.3 Development

Due to the size of this small-scale sprint to test the available technologies, the development was quick and provided little issue. To test these technologies, an application was developed to detect the eight buttons on my Launch Control and simulate a unique key press for each button. This was easily achieved using the Windows Forms simple "SendKeys" class.

Functionally, the application would work by reading the event data from a DryWetMIDI receive MIDI controller signal event, parsing that data, and reacting to that data using a list of if statements for each button.

At this point, I still wasn’t sure why the MIDI controller listener would occasionally stop listening. However, I had already theorised how to fix this issue. My initial web searches

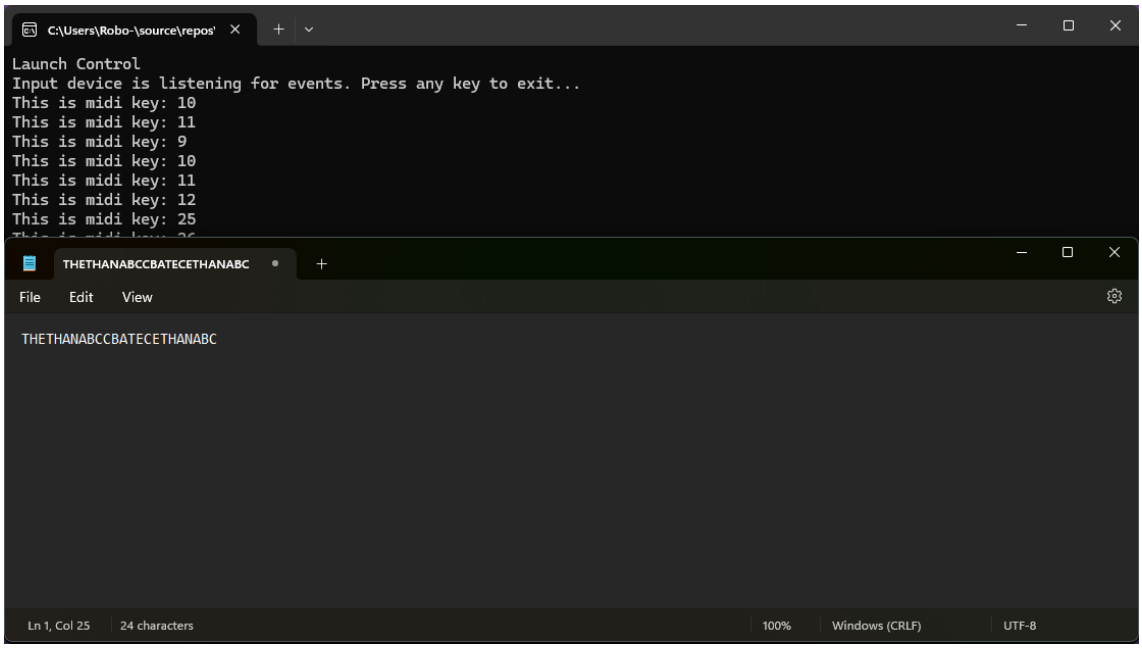


Figure 6.1: Iteration One After Sending MIDI Signals to a Notepad

to identify this issue and its possible fixes were fruitless. Thus, I theorised that to fix this issue, an asynchronous thread should check if the listener is still active at a rapid interval and, if not, add the listener.

6.3 Developing the GUI (Iteration Two)

This iteration aimed to implement most of the functional requirements into a GUI. This would allow testing the current planned GUI design while learning to implement the more necessary features.

The main window and project file were generated using Microsoft Visual Studio's "WPF Application" template to initialise the project. Some design alterations were made to the main window itself after this creation. The window was kept at the default 450 by 800 pixel size, and other parameters were altered, such as the "ResizeMode" being set to "NoResize" to prevent any window size changes and the "ShowsNavigationUI" being set to "False" to hide any unneeded preset UI overlay that comes with the template. All these changes were made using ".xaml", the standard markup language for WPF.

Using the Project toolbar in Visual Studio, a new "UI" page was created to house all the UI markup and C# logic. WPF has two main types of UI elements: Windows and Pages. Windows are completely separate windows that, although they can hold markup, reference pages that hold UI markup. To make "MainWindow.xaml" load "UI.xaml" when the application starts, adding "Source='UI.xaml'" to the MainWindow's parameters will do this.

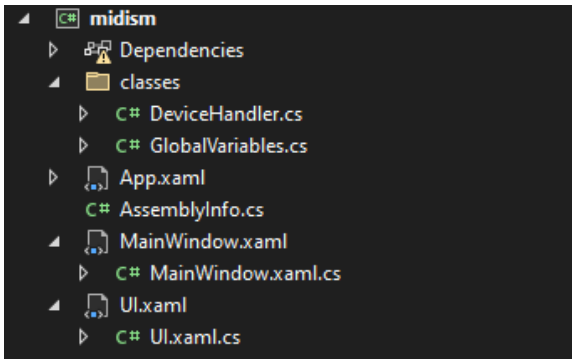


Figure 6.2: Solution Explorer View of Files in Sprint Two Project

6.3.1 Classes

Two main classes were added using the Project toolbar: DeviceHandler and GlobalVariables.

GlobalVariables contains all data that must be accessed across the application. Two variables are stored here: "activeListeners" and "translations." The variable "activeListeners" is a string list containing the names of all current devices connected with a listener attached; this is used to check if a listener has already been added to a device. The variable "translations" is a Dictionary with the index being an integer and the value being a string list; this holds all current translations set by the UI.

DeviceHandler contains all the functions necessary for communicating with MIDI controllers, functions like getting all of the current connected MIDI controllers, adding a signal listener to the MIDI controller, and the primary reaction function, which fires every time a signal is received. Most of these functions revolve around the DryWetMIDI package; however, the core function of the class is the "OnEventRecieved" function. This function first parses the event data into a more processable integer format, taking the MIDI signal number and activation level number, a 0 to 127 integer representing how activated a MIDI signal is. Before any translation or reaction occurs, the function checks if the MIDI signal number is in the translation list located in the GlobalVariables class. When the MIDI signal is confirmed, the reaction is activated.

6.3.2 UI Design

Two grids were used to replicate the UI design planned in the design section. One grid held the top bar buttons, and the other held the translations and their parameters. The top bar grid consists of one row with multiple columns, some containing buttons to control the software. The translation grid is empty when it loads but becomes filled after using the add translation button. The translation grid has more columns than the top bar grid to hold the different parameters. Additionally, the translation grid has altered parameters to display a vertical scroll bar when the contents of the grid expand further than the window's

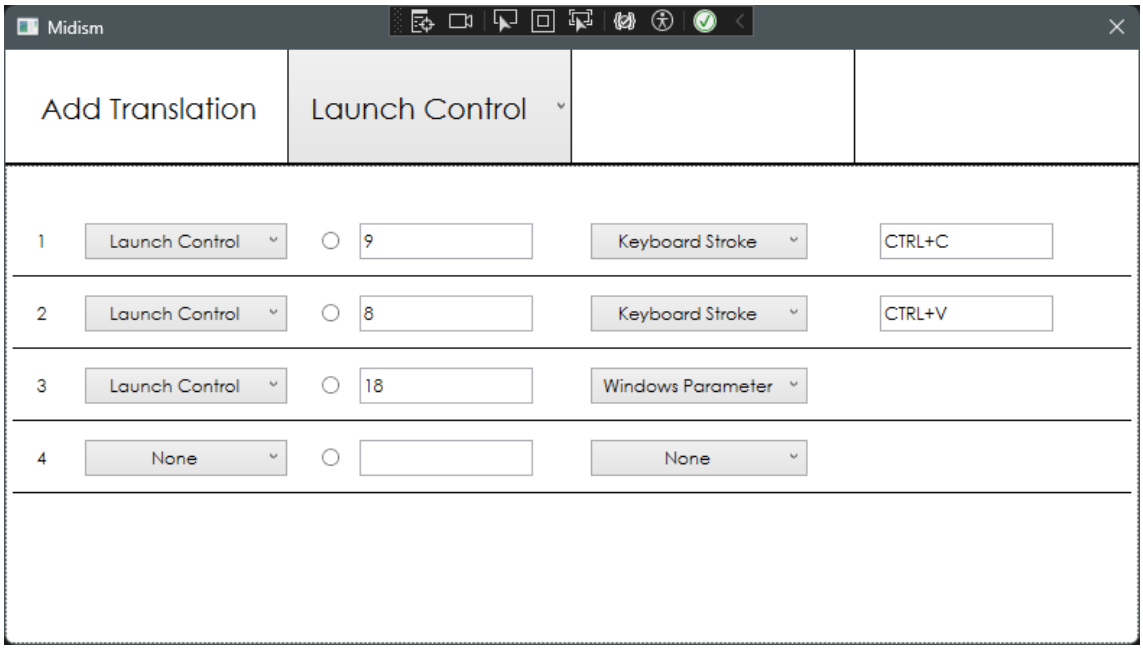


Figure 6.3: Solution Explorer View of Files in Sprint Two Project

width.

6.3.3 UI Logic

The code behind the application's UI has many small niche features to help with the general usability of the system; this section will cover both these small, ease-of-use features and the large, more interesting ones. The order of these features is chronological when they would be used.

When the application is first loaded, the device selector drop-down is filled with the option "None." This none option prohibits adding translations while no active MIDI controllers are connected and selected. When the drop-down is opened, the application checks for any MIDI controllers and adds them to the drop-down items. If previously connected devices are disconnected, they will be removed from this list. This should prevent any potential input errors.

When a new translation is added, it will default to whichever device is selected in the device selector drop-down; however, changing the MIDI controller from which the translation will react to is still available using the first drop-down. The second input box, a text box, is the input for which the translation should react to the MIDI signal number. This text box is set only to accept numbers; any attempt to input numbers isn't added to the text box. Temporarily, for iteration two, the MIDI signal number for each button or dial for the MIDI controller can be found in the debug console for Visual Studio after pressing said input.

The third input is a drop-down box containing the two current options; selecting either option will update the fourth input's type. For example, when "Keyboard Stroke" is selected, a text box will appear indicating which keyboard inputs it should send.

The fourth input is currently only limited to keyboard inputs for this iteration. The "SendKey" function inside Windows Forms has specific parameters that allow the use of "Control", "Shift", and "Alt" key modifiers. Special processing is applied to the key press input to translate text to keyboard input and to add specific modifiers. If "CTRL+" is detected in the fourth input, the "Control" modifier is applied to the "KeySend" function; the same goes for "SHIFT+" and "ALT+". This functionality allows for the activation of keyboard shortcuts.

6.3.4 Development Issues

During development, a few issues were uncovered. Drop-down boxes, buttons that expand into multiple options the user can select, are called "Combo Boxes" in WPF. Combo boxes, by default, are coloured light grey. To match iteration two's current work-in-progress white box aesthetic, countless attempts were made to change combo boxes from grey to white. It was not necessarily essential to alter the current aesthetic of the application. Still, the attempts were to prove whether combo boxes' colour could be changed for a later aesthetic change. To alter an element's colour, using the element's parameters, such as "backgroundColor", can typically change the element's colour. The combo box is a rare exception with useless parameters. After further investigation, it was discovered that the style of all combo boxes must be altered using "Application.Resources" in the elements .xaml file. This alteration involves completely recreating the combo box; thus, it felt like a job to be done in the third iteration.

The most significant issue faced in this project's iteration was the combination of the WPF and Windows Forms. As previously mentioned, Windows Forms is a depreciated version of WPF; however, it causes many issues if both are included in the same project. Windows Forms is the only way to add keyboard press functionality with modifiers, excluding the existing over-engineered packages found on Nuget. Previous versions of Microsoft Visual Studio had a feature called "Framework Referencer" which was a GUI window which allowed easy selection of specific classes from Frameworks, such as Windows Forms. However, this feature does not exist in Visual Studio 2022. Therefore, the apparent only way to obtain Windows Forms functions is to include Windows Forms in the project.

Windows Forms and WPF can be added in the ".csproj" file using the "<UseWPF>" or "<UseWindowsForms>" tag. However, if both are added simultaneously, any reference to either framework confuses each other, and the compiler does not know which framework to use, rendering both unusable. The workaround that is not popular information to any programming forum is to use the "<FrameworkReference>" tag in the project file, which

acts as a manual way to perform the now deprecated "Framework Referencer". This method adds Windows Forms as a separate package that can be referred to and does not change how the application is compiled.

Finally, the next sprint must consider how reactions are activated. Currently, a reaction will occur when any signal comes through when a user presses and releases a button that counts as two messages, thus activating a reaction twice. This next sprint must ensure activation numbers are accounted for.

6.4 Refining the Application (Iteration Three)

This final sprint aims to make the application aesthetically pleasing by implementing the colour scheme created in the design section, adding additional features, and cleaning how features work.

6.4.1 Application Aesthetics

During this sprint, XAML and its styling issues became immediately apparent when trying to recolour the application. After numerous attempts to use properties of elements to change their visual style, how to alter more specific parts of elements became a mystery. A solution was found after researching the topic on forums and searching the .NET documentation. XAML's "ControlTemplate" tag changed the element style dramatically. Using the template found on .NET documentation website (Microsoft, 2006), which was simply a version of a default style of the element of choice, the template was edited to fit the visual style planned in the design section.

The significant change to the style was the colour. However, some additional alterations were made to other elements of the application. Combo boxes were changed heavily as they are made of several parts. A combo box is made of three parts, "SelectedBox", "ToggleBox", and "ContentBox". Each must be individually coloured and needs content location adjustment using "margins" to fine-tune exact distances between content and borders. Both the "Add Translation" button and "Controller Selector" combo box have an "OnHover" effect, which shows the user that these elements can be interacted with. Additionally, a top border was placed above both elements to guide users into understanding that they can be interacted with.

Finally, a row underneath the top bar was added to show the different translation parameters. Although a minor feature, this could massively help user learnability as users can immediately understand the anatomy of a translation. The red circle and question mark in Figure 6.4 are made from one textbox composed of two "<Run>" tags, allowing for easy formatting of both symbols. The circle is a Unicode (2024) character taken from their website coloured red using this "<Run>" technique.

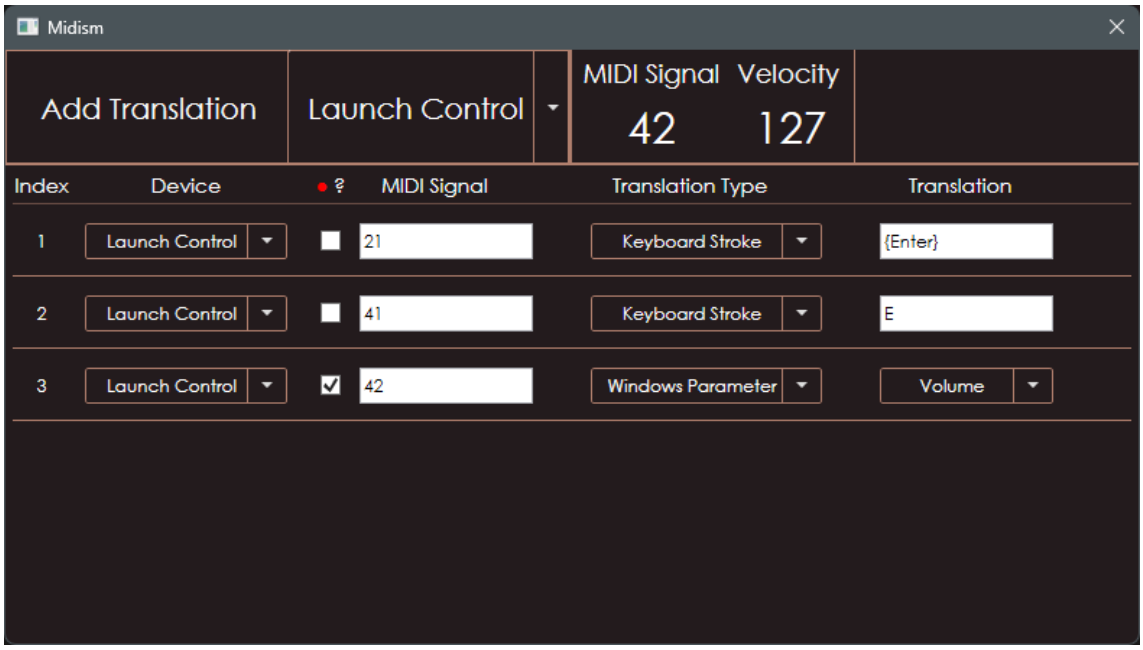


Figure 6.4: Final view of the application

6.4.2 Additional Features

Three features were added in this sprint: "Signal Viewer," "MIDI Recorder," and "Windows Volume Control." Signal Viewer allows users to easily comprehend what the application sees the connected MIDI controller doing. This small window is easily viewable and shows the last signal received from the selected MIDI controller and the signal's velocity.

The MIDI Recorder is a check box next to every MIDI signal selection box, so any new MIDI signal that comes through is immediately placed inside that box. This removes an extra step from creating translations and makes the application quicker.

Windows Volume Control is a feature which controls the volume of the default Windows playback device based on the velocity of the selected MIDI signal. Primarily used for knobs and potentiometers, this will allow quick and expressive control over volume. A unique script was used to control the volume of the Windows operating system. Although a Nuget package would be used regularly, after a thorough search through Nuget, no package with that functionality was found. Fortunately, Meinz (2023) released an open-source C# script for getting and setting the volume for the default Windows audio device.

6.4.3 Cleaning the application

To prevent errors and provide better project usability, minor fixes were made across the application to improve the experience. Namely, the way "Keyboard Stroke" translations reaction to velocity has changed; instead of "Keyboard Stroke" being fired to every MIDI signal, it now only accepts MIDI signals with a velocity above one. This alteration stops

double firing when a MIDI button is unpressed, allowing dials and potentiometers to use the Keystroke feature.

6.5 Application Guide

To assist with the usability of this application, a "README.md" will be used alongside it. Although using said document wouldn't be imperative for the application, it includes essential insight to guide a starting user. This guide ensures excellent aesthetic and readability by utilising the advanced syntax of "Markdown," a standard markup language used to format text documents. A copy of said document can be found in Appendix D.

6.6 Conclusion

The application was successfully developed with minor, yet easily overcome, problems. Alongside the application, a "README" was written to support the use of this application. The small problems faced were about the MIDI controller signal listeners and the visual aesthetics of using XAML.

Chapter 7

Testing

7.1 Introduction

This chapter explains and analyses how testing was implemented into the project, covering how the software was tested at a development level and usability at a user level. The usability test involved created a test plan, refining the plan after the first test then analysing the results of the tests.

7.2 Software Testing

Using the debugging mode in Microsoft Visual Studio was imperative to development and was used until the application's final release build. The debugging tool identified ongoing and potential issues and showed how to fix them. These tools helped with error prevention and data validation.

Although Microsoft Visual Studio contained features that allowed unit tests to be generated from code files, Due to the small scale of the application, it was decided that it would not be implemented due to time constraints and a lack of need based on limited user input. The code for this application was written to prevent the application from breaking if the wrong data was input into a function.

7.3 User Test

To test the usability of this artefact, a think-aloud test was performed on a set of test participants with varying technological skill levels. This test is essential as it can identify potential usability issues. Using participants with varying skill levels for technology and music can help find learnability issues. Participants with solid skills may have insight into creating more usable UI, and lesser-skilled participants may outline some user frustrations.

7.3.1 Test Plan

This test aimed to gather feedback and opinions on the artefact, use the results, and then make changes to the artefact. A think-aloud test was more suitable for this aim as the results of think-aloud tests are the raw thoughts of a user. A think-aloud test involves a participant performing a complex task and saying their thoughts aloud (Nielsen, 2024a). Not talking or influencing the participant in any way will help identify and struggle points of the application.

The data was collected by recording the audio from the test for further analysis and exact quotation, alongside a notepad to write down observations about the test. The only data recorded from the test was observations about the test, and any raw data was permanently deleted upon the final submission of this report.

The testing environment was the control variable of the test. The testing environment was where the average user began when using the application for the first time. A MIDI controller with knobs was provided. The MIDI controller was disconnected, and the computer was set up with only the application freshly launched, an audio player of a song, and a digital notepad. This was the environment set up for every participant. After the participant read the information sheet and signed the study consent sheet, the participant was ready to participate. The participants were given two objectives, "Make any button on the device output the word 'Hello'" and "Make a dial on the device control the computer volume". During the test, the moderator would not say anything to help the participant or guide them to reach the objective.

After the test, two questions were asked: "Were there any points of frustration whilst using this application?" and "Can you think of any features that could be added to aid your satisfaction with the application?" These two questions aimed to outline any issues and areas for improvement clearly.

The participants were peers and people I knew personally, so I could more accurately identify their skill level.

7.3.2 Test Revision

Immediately during the first test, the vital flaws of the test were shown. Having the MIDI controller disconnected from the computer was supposed to help guide new users in separating the device from the computer; however, after seeing this in action, it immediately seemed redundant and would only prove to confuse lesser-skilled users.

The test objectives were not as straightforward as they needed to be for a lesser-skilled user; seeing the initial confusion of the first participant, the test objective was expanded further. The new test objective given to participants, spoken verbatim, is "You have two objectives: make the button marked '1' keystroke the word 'hello' in the notepad and

control the volume of the song playing using the knob above the first button. Use the 'MidiMeld' application to assign these actions to this device." This made the participant's objective far more explicit.

7.3.3 Results

The results varied heavily according to the participants' technology skill level. The test was performed on a total of five participants, which was enough to show a large room for improvement. While the new objective statement helped participants massively, it also outlined massive design flaws in the application's usability.

Initially, translations couldn't be added to increase usability unless a device was selected. This backfired immediately, as participants attempted to add translations without understanding that a device needed to be selected. This was an obvious point of confusion and frustration. Removing this "safe-guarding" feature would eliminate this confusion and increase the usability and speed of learnability for users.

Some participants felt the "Record MIDI" feature was confusing, as a checkbox, participants would click the record MIDI button on a translation and forget to unchecked the box. Perhaps if the checkbox unchecked itself once a MIDI signal is recorded, this would remove that potential confusion and error.

Participants without previous knowledge about MIDI and how MIDI works struggled massively understanding how the MIDI controller was reacting, however, previous MIDI users immediately understood the MIDI signal viewer panel and how to use the "MIDI Signal Selection". Lesser-skilled participants suggested if they knew more about MIDI, it would be far easier to understand the process of using the application.

Surprisingly, all participants found the application very understandable after adding a translation, adding a MIDI signal to that translation and then clicking the "translation type" combo box. Almost immediately, participants could easily replicate what they did. From analysing the audio from the think-aloud, all participants understood the groupings of actions and types upon pressing the "translation type" combo box.

None of the participants used the MIDI signal viewer window to add a MIDI signal to a translation, which proved attractive because it showed how the MIDI controller was reacting. All participants would use the record button to assign MIDI signals. However, after moving parameters on the MIDI controller, most participants saw the MIDI signal panel adjust to their actions. They enjoyed seeing how the MIDI controller reacted, especially the "velocity" change for the dials.

7.3.4 Test Analysis

After doing this study, it became clear that, although choosing a think-aloud method for this test was a good idea, it did have some flaws. Some participants found it incredibly

easy to talk nonstop without a filter about their choices, while others occasionally froze and thought internally. Reminding participants that this was a think-aloud test in a friendly tone returned them to a "think-aloud" state.

This test massively outlined glaring issues with the design of the application. This concludes that the study was successful, but perhaps more tests, different types of tests and a wider audience of participants with further variants of skill levels would have garnered more data to analyse. Regardless, valuable lessons were learnt about the developed UI.

Initially, there was some doubt about testing at this application stage. To save time, low-fidelity applications are tested to show early problems; however, this application was highly high-fidelity. I believe testing at this stage had shown far more valuable data about how users would use this application instead of how they might.

7.4 Conclusion

After the first attempt, the testing plan needed alterations; however, the subsequent tests effectively underlined the application's vital usability flaws. The software testing implemented was mainly through the use of Microsoft Visual Studios testing features which identified potential logical issues with the code. The usability tests performed were effective but needed more tests to get more data about the overall usability of the application.

Chapter 8

Evaluation

8.1 Introduction

This chapter will evaluate whether the requirements have been satisfied and analyse the planned methodology. Both functional and non-functional requirements were evaluated, and their implementation is explained.

8.2 Evaluation Approach

The approach for evaluating the requirements was to assess whether the requirements had been met by analysing the project artefact's ability to perform the individual requirements. Evaluating the methodology involved comparing the planned methodology to how the methodology performed.

8.3 Functional Requirements

ID	Requirement Met?	MoSCoW	Explanation
01	Yes	Must	The application can read and send MIDI signals for any MIDI controller detectable by Windows 11.
02	Yes	Must	The application can perform different actions based on a specific MIDI signal.
03	Yes	Must	Translations can be set to receive signals from a specific device.
04	Yes	Must	Translations can be set to control Windows OS Volume.
05	Yes	Must	Translations can be set to simulate keyboard input.
06	Yes	Must	Translations can be set to react to specific signals.
07	Yes	Should	MIDI signals from the selected MIDI controller can be seen at the top bar of the application.
08	Yes	Should	MIDI signals can be set to a translation using the "record" checkbox next to each translation's MIDI signal input box.
09	No	Should	MIDI to MIDI did not seem necessary nor possible to test, as sending MIDI back to the MIDI devices in my possession would not show anything.
10	No	Could	Script execution, whilst possible, could not be implemented due to time constraints. If implemented, it would have been the third type of translation.
11	No	Could	Background execution of this application is possible; however, it could not be implemented due to time constraints.
12	No	Could	Velocity Specific Translation Activation was not attempted due to time constraints, although possible.

Table 8.1: Evaluated Functional Requirements

8.4 Non-Functional Requirements

ID	Requirement Met?	MoSCoW	Explanation
01	Partially	Must	During the usability test, some participants found it easier to learn the application's design; however, others struggled. More testing data would be necessary to confirm.
02	Unknown	Must	As "Aesthetically pleasing design" is a subjective opinion, this is a difficult requirement to measure. More testing would be necessary to identify this.
03	Yes	Should	All code to the project artefact is licensed under the GNU GPLv3 license and hosted on GitHub.
04	Partially	Should	A help guide was made to be used alongside the application; however, there is no built-in help feature.

Table 8.2: Evaluated Non-Functional Requirements

8.5 Methodology

Agile was a highly effective software development methodology for this project. Splitting the project into three parts helped me massively understand the step-by-step development process compared to tackling the project simultaneously. Although this implementation of Agile wasn't a traditional version of Agile, it served as an excellent solo development methodology. An alteration that would have made this implementation better was if a usability test was performed after every sprint to gather input about the design.

8.6 Project Planning

Despite the planning, the time management for this project was lacking heavily. There was a significant amount of time slippage where other responsibilities leaked into day-to-day life when developing this project. It was pretty naive to assume there wouldn't be time slippage; however, the time management issues mainly stemmed from mental blocks.

Additionally, the Kanban board proved helpful during the early stages of development but was abandoned during the late half of the second sprint. As we relearned C# and WPF in the second sprint, programming sessions became less about completing the individual tasks of the sprint and more about simply finishing the sprint. Kanban would be deemed

more suitable for larger-scale projects with lots more moving parts; however, this project's features didn't require too much code and thus didn't warrant a Kanban board.

8.7 Conclusion

In this evaluation, it was discovered that the project was mainly successful in meeting the requirements. The planned methodology was the correct choice for this project; however, time management with the Gantt chart and Kanban was fruitless.

Chapter 9

Conclusion

9.1 Project Aims

This project aimed to research MIDI and usability, identify requirements through existing solutions and develop an "easy-to-use" application that replicates the existing solutions' functionality. All three of these aims were achieved successfully, excluding the fact that the usability had shown where the application was not easy to use.

Initially, the research process halted this project due to the incorrect idea that the entire literature review had to be completed before any other step was to take place. Project research is a continuous process that never finishes until the project does. This mental block for the project was removed once different productive strategies were placed. The price that this mental block had was a large chunk of time allocated to this project, thus removing the potential for a final sprint or more features being added to the project.

The research on MIDI and usability was generally successful, as good information was found about both topics. However, the usability part of the project artefact may have benefited from more research in the area instead of light research on specific topics.

Requirements were adequately identified using other existing solutions, which also gave inspiration for more unique and creative ideas. Regrettably, these ideas were not implemented due to time constraints.

While the application was complex for some, it effectively performed the basic "MIDI Translation Application" features. Some features, such as the "Remove Translation" feature, should have been considered but were wholly forgotten. During the usability testing, it became highly apparent that although the add button was added, the remove button was not. It is not incredibly important for an application that needs far more work to be considered on par with existing solutions; however, it was an obvious feature in hindsight.

9.2 Limitations

9.2.1 Project Management

The main limitation of this project was management. Lacking experience in a project of this scale, project management skills were lacking, and time resources were seriously hindered. As mentioned previously, the mental block with the literature review consumed much of the planned time allocated to this project. Alongside additional university responsibilities such as other modules and coursework, there was less time to work on this project than planned. Poor personal time management seriously affected this project as a whole.

9.2.2 Technical

There were a few technical limitations to this project. The two MIDI controllers used for this application, the "Launch Control" and the additional "Launchpad Mk2", served as excellent testing devices. MIDI, as a whole, functioned perfectly. However, some programming libraries require unique workarounds to make them function, such as the issue in Chapter 6.2.3, where the MIDI listener suddenly stops working.

9.3 Future Work

There is a substantial amount to be done on this project. This development process has severely revealed the critical aspects of this project, its flaws, and how to succeed.

9.3.1 Design

Although this application's design fulfilled usability, it severely lacked software development philosophies, such as maintainability. The design was not designed to include more advanced features that would include using more UI space. If different translation types were added, it would be simple to add them to the translation type combo box, but if the way translations were activated were to be changed and enhanced, the design could not fit such a complex system.

If changes were to be made to the design, they should be to make it scaleable. A worst-case scenario for this application would be if more features were added, cramping the current design style, making the application confusing and chaotic, and removing the "Minimalist Design" researched in the literature review.

9.3.2 Feature Ideas

Script Execution was a semi-frequent feature in the existing solutions. It inspired this project when I tried to make personal MIDI controllers perform complex tasks such as running executable applications. This feature could be added to the translation type combo box very simply and have an input box specifying the location of the executable.

Velocity-based translation activation was an idea seen in no other existing solution, mainly focusing on MIDI dials. Dials are not implemented enough into the existing software; thus, it would be an exciting and unique idea for more customisable control over the application. Expanding on this idea further, where the completion of a dial turn is based on a MIDI signal velocity, different velocities would activate different translations depending on the velocity.

Controlling Windows 11 volume is a unique feature, but it lacks customisation. To improve this feature, perhaps instead of controlling a device's volume, controlling individual program volumes would be a better idea than turning down VoIP software or music software.

Lastly, an issue became apparent whilst testing: the application would dramatically slow down if a translation were assigned two actions to one MIDI signal. Whether this is a build issue or an issue with another package remains unknown; however, it would be best to fix this issue by allowing multiple translations for a singular signal.

9.4 Reflection

This development was highly eye-opening to the world of music software development and has furthered my interest in this project and this software sector. I have gained massive insight into working with MIDI and C#, and although pre-existing knowledge did exist, this project has only expanded my interest in these topics.

Given more time, this project would have been far better in quality. However, stopping at sprint three to perform usability tests stopped potential issues from growing and becoming a serious design flaw. If the project continues after this report, the whole design will be reconceptualised to ensure the long-term maintainability of open-source projects. The logical code developed in this project is excellent groundwork for a potentially prosperous future personal project.

Bibliography

Alsaqqa, S., Sawalha, S., & Abdel-Nabi, H. (2020). Agile software development: Methodologies and trends. *International Journal of Interactive Mobile Technologies (iJIM)*, *14*(11), 246. https://doi.org/10.3991/ijim.v14i11.13269

AWS. (2024, May). What is devops? https://aws.amazon.com/devops/what-is-devops/

Bome Software. (2019, September). Midi translator classic. https://www.bome.com/products/mtclassic

Clegg, D., & Barker, R. (1994). *Case method fast-track: A rad approach*. Addison-Wesley Longman Publishing Co., Inc. https://doi.org/10.5555/561543

Draw.io. (2024, May). Draw.io: Security-first diagramming for teams. https://www.drawio.com/

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). Devops. *IEEE Software*, *33*(3), 94–100. https://doi.org/10.1109/ms.2016.68

Fessenden, T. (2022, February). Aesthetic and minimalist design (usability heuristic 8). https://www.nngroup.com/articles/aesthetic-minimalist-design/

Gibson, J. (2016). The midi standard: Introduction to midi and computer music. https://cecm.indiana.edu/361/midi.html

Meinz, M. (2023, April). Getting and setting windows system audio volume. https://www.codeproject.com/Tips/5358341/Getting-and-Setting-Windows-System-Audio-Volume

Melanchall. (2024, May). Melanchall/drywetmidi: .net library to read, write, process midi files and to work with midi devices. https://github.com/melanchall/drywetmidi

Microsoft. (2006, March). Controltemplate examples. https://learn.microsoft.com/en-us/previous-versions/dotnet/netframework-3.0/aa970773(v=vs.85)

Mitchell, L. (2022, December). Midi: Your guide to midi and midi controllers. https://blog.native-instruments.com/guide-to-midi-and-midi-controllers/

Mozilla Developer Network. (2024, May). Web midi api - web apis: Mdn. https://developer.mozilla.org/en-US/docs/Web/API/Web_MIDI_API

Nielsen, J. (2024a, January). Thinking aloud: The #1 usability tool. https://www.nngroup.com/articles/thinking-aloud-the-1-usability-tool/

Nielsen, J. (2024b, January). Usability 101: Introduction to usability. https://www.nngroup.com/articles/usability-101-introduction-to-usability/

Radigan, D. (2024, May). Kanban - a brief introduction. https://www.atlassian.com/agile/kanban

Rowoldt, S. (2024, May). Midikey2key. https://midikey2key.de/

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, *35*(3), 8–13. https://doi.org/10.1145/1764810.1764814

Shneiderman, B. (2016, May). The eight golden rules of interface design. https://www.cs.umd.edu/users/ben/goldenrules.html

Unicode. (2024, April). Unicode geometric shapes. https://www.unicode.org/charts/nameslist/n_25A0.html

Vandenneucker, D. (2024). Midi tutorial for programmers. https://www.cs.cmu.edu/~music/cmsip/readings/MIDI%20tutorial%20for%20programmers.html

Appendix A

Project Initiation Document

School of Computing
Final Year Engineering Project

Project Initiation Document

Ethan Egerton

BSc Software Engineering

Musical Instrument Digital Interface Signal Manipulation

1. Basic details

Student name:	Ethan Egerton
Draft project title:	Musical Instrument Digital Interface Signal Manipulation
Course and year:	BSc Software Engineering 3rd Year
Project supervisor:	Alexander Gegov

2. Degree suitability

My project is relevant to my course, software engineering, because I will be developing an application that can read Musical Instrument Digital Interface (MIDI) signals and change the outputs. This will require the development of software capable of reading MIDI inputs, and activating different signals or actions based on the value of the signal. Although I do not have experience in working with MIDI on a programming level, I have culminated experience in programming in languages that have MIDI manipulation libraries or capabilities.

3. The project environment and problem to be solved

This project, at a basic level, is the manipulation and the control over MIDI controllers. A MIDI controller is a device that can send MIDI signals to a computer, where the computer can interpret said signals and produce an output. Normally these outputs would be musical notes played from a Digital Audio Workstation (DAW), however I plan for my application to be able to trigger keystrokes and execute scripts.

The problem with MIDI controllers is that they often are limited to simply sending their designated MIDI signals; there is no built-in way to change or alter the actions of a MIDI controller. My project benefits any user of any MIDI controller, by extending the use of the controller by allowing customisable outputs.

The MIDI Controller industry is ever growing, as is the industry that uses MIDI controllers the most, the music industry. The music industry is ever growing as digital music playback services, such as Spotify and SoundCloud become increasingly accessible to wider audiences. Music Producers require MIDI Controllers, from Keyboards to Pads (A MIDI device with rubber tipped activators primarily used to play digital drums using your fingers), to streamline the music creation process.

4. Project aim and objectives

Aim:
The overall aim of the project is to create a software that can make MIDI controllers useful in more than one area, the current area being music creation.

Objectives:

- Be able to create a software that can react to a MIDI input from a MIDI controller.
- Be able to react to two different MIDI controllers at the same time, each with different configurations.
- Make the application a GUI which is pleasing and easy to use.
- Manipulation of MIDI inputs should be low latency.
- Software should be easily deployable to Windows machines.

5. Project constraints

The main constraint I will be facing is time, with the final deadline being the 3rd of May, I'm not given an extreme amount of time to complete the project. I am not working with a client so I am my only constraint to the project. MIDI controllers are not a dying industry so I am in no rush to make this project

6. Facilities and resources

To reach my aim and objectives, I will require resources to help test and develop my project. Among these resources will be two or more MIDI controllers, to prove that my application can react to two or more controllers at the same time. I will obviously require a laptop or PC to develop, deploy and test the software. All these resources are currently in my possession.

7. Log of risks

Description	Impact	Mitigation/Avoidance
Workstation Failure	Heavy time delay as I would be forced to work at the library	Back up work to GitHub and Google Drive, and work on University campus workstations
MIDI Controller Failure	Heavy time delay as testing would rely on using MIDI controllers	Have a small budget available to buy a cheap and basic MIDI controller if need be, otherwise, ask University if it would be possible to lend or use a MIDI controller for testing
Loss of Internet	Stops me from accessing an online copy of my project documentation or	Try to make all my work available offline, however, if not, work on campus

	a live version of my project	
Illness (short)	A few days of disruption	Keep ahead of schedule and be prepared to continue work after disruption
Illness (long)	Many days or weeks of disruption	File for ECF (extenuating circumstance form) and keep in contact with my supervisor often
Bereavements	Few days or weeks off	File for ECF (extenuating circumstance form) and keep in contact with my supervisor often. Attempt to be ahead of schedule to mitigate days lost

8. Project deliverables

The artefact to be developed is an application, that can be easily deployed, and that fulfils the objectives, such as being able to change MIDI signals. However, heavy documentation should be included, such as test logs, programming documentation, ERDs (if needed), UML diagrams, user guides, requirements.

9. Project approach

I will need to research whether this has been done before, and whether it's been successful or effective. If there are applications that have been made for this problem, that’s good because I can use each application's pros and cons to understand how I should develop the application.

I will have to learn more in depth about MIDI, how it works exactly and how to manipulate it. There are a range of different books within the library that describe deep guides on how to program with MIDI, this information will be the groundwork for this project.

Using past projects and my own research, I can identify the best tools and programming languages to use for this project. My current idea would be to develop in C#, as not only do I have experience in the language, but the language has built in audio and MIDI control as well as a rich community of external libraries which I could implement into my application to reduce the workload of building this application. However, this is all subject to change once I have enacted enough research to produce an in-depth idea and strategy to develop this application.

10. Project tasks and timescales

No	Stage	Dates	Main Tasks
1	Project Initiation	18/09/23 - 20/10/23	Choose topic and supervisor, write PID

2	Requirement Gathering	21/10/23 - 08/12/23	Research and study the topic area, complete ethics review
3	Development	09/12/23 - 09/03/24	Create the artefact and the documentation for it
4	Testing	07/11/23 - 04/04/24	Test existing applications, test during development and test final product
5	Report Writing	01/11/23 - 17/05/24	Write and combine reports to create the final report

11. Supervisor meetings

Supervisor meetings will be weekly, however more details involving my supervisor's time and availability will be available after the first reading week (30th of October). My supervisor and I are quite flexible and are available for both online and in person meetings, however, in-person meetings are often ideal but sometimes I will need to share something from my computer, therefore an online meeting would be better. Our choice of contact is both email and mobile phone messaging as both are a quick method of communication and can be interchangeable based on our current availability situations.

12. Legal, ethical, professional, social issues

I do not plan for this project to collect any data at all, therefore being GDPR compliant will be very easy. I will try to make my software as safe as possible, not accessing any data I don't need to. I am simply developing software that MIDI controllers will interact with, I am by no means altering MIDI controllers. I do not see any ethical, professional, or social issues that my project could cause.

13. Permission

Please tick

☑ I give permission for my PID to be made available to other students as examples of previous work.

☐ I do not give permission for my PID to be made available to other students as examples of previous work.

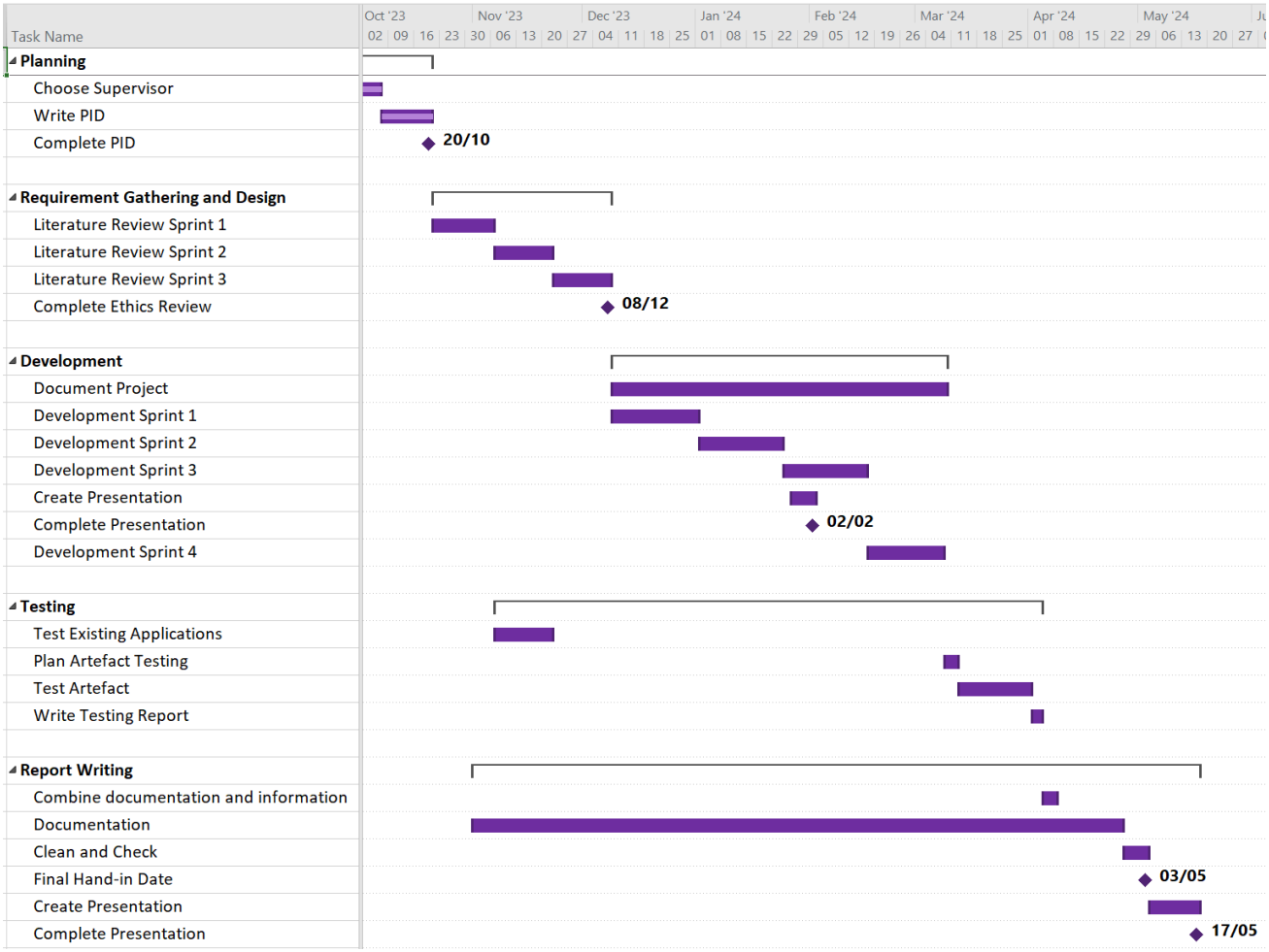
Date: 18/10/2023

Supervisor: Alexander Gegov

Supervisor Signature:

Appendix A: Gantt chart

.



Appendix B

Ethics Certificate

Certificate of Ethics Review

Project title: Extending Computer Interactivity Using Digital Musical Devices

Name:	Ethan Egerton	**User ID**:	up2067147	**Application date**:	06/03/2024 03:13:41	**ER Number**:	TETHIC-2024-108215

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are **Elisavet Andrikopoulou, Kirsten Smith**

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **School of Computing**
What is your primary role at the University?: **Undergraduate Student**
What is the name of the member of staff who is responsible for supervising your project?: **Alexander Gegov**
Is the study likely to involve human subjects (observation) or participants?: Yes
Will you gather data about people (e.g. socio-economic, clinical, psychological, biological)?: No
Will your data collection be strictly limited to gathering anonymous insights about a particular artefact or research question (e.g. opinions, feedback)?: Yes
Confirm whether and explain how you will use participant information sheets and apply informed consent.: I will use a participant information sheet to fully explain what data will be collected from the user and what the project and study is about. I will use an consent form to ensure written consent is gathered from the participant. I will use the university templates for the mentioned documents.
Confirm whether and explain how you will maintain participant anonymity and confidentiality of data collected: I will maintain participant anonymity by not collecting any personal data that could link the data gathered from the study to the participant. The only data gathered from the study will be the results of the study, nothing more. If any personal data is necessary to be gather from the study, it will be anonymised to ensure nothing can link the participant to the study. All data will be promptly destroyed after the project is complete, submitted and the deadline is over.
Will the study involve National Health Service patients or staff?: No
Do human participants/subjects take part in studies without their knowledge/consent at the time, or will deception of any sort be involved? (e.g. covert observation of people, especially if in a non-public place): No
Will you collect or analyse personally identifiable information about anyone or monitor their communications or on-line activities without their explicit consent?: No
Does the study involve participants who are unable to give informed consent or are in a dependent position (e.g. children, people with learning disabilities, unconscious patients, Portsmouth University students)?: No
Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?: No
Will blood or tissue samples be obtained from participants?: No
Is pain or more than mild discomfort likely to result from the study?: No
Could the study induce psychological stress or anxiety in participants or third parties?: No
Will the study involve prolonged or repetitive testing?: No
Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No
Are there risks of significant damage to physical and/or ecological environmental features?: No

Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No
Does the project involve animals in any way?: No
Could the research outputs potentially be harmful to third parties?: No
Could your research/artefact be adapted and be misused?: No
Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: No
Please read and confirm that you agree with the following statements: I confirm that I have considered the implications for data collection and use, taking into consideration legal requirements (UK GDPR, Data Protection Act 2018 etc.), I confirm that I have considered the impact of this work and and taken any reasonable action to mitigate potential misuse of the project outputs, I confirm that I will act ethically and honestly throughout this project

Supervisor Review

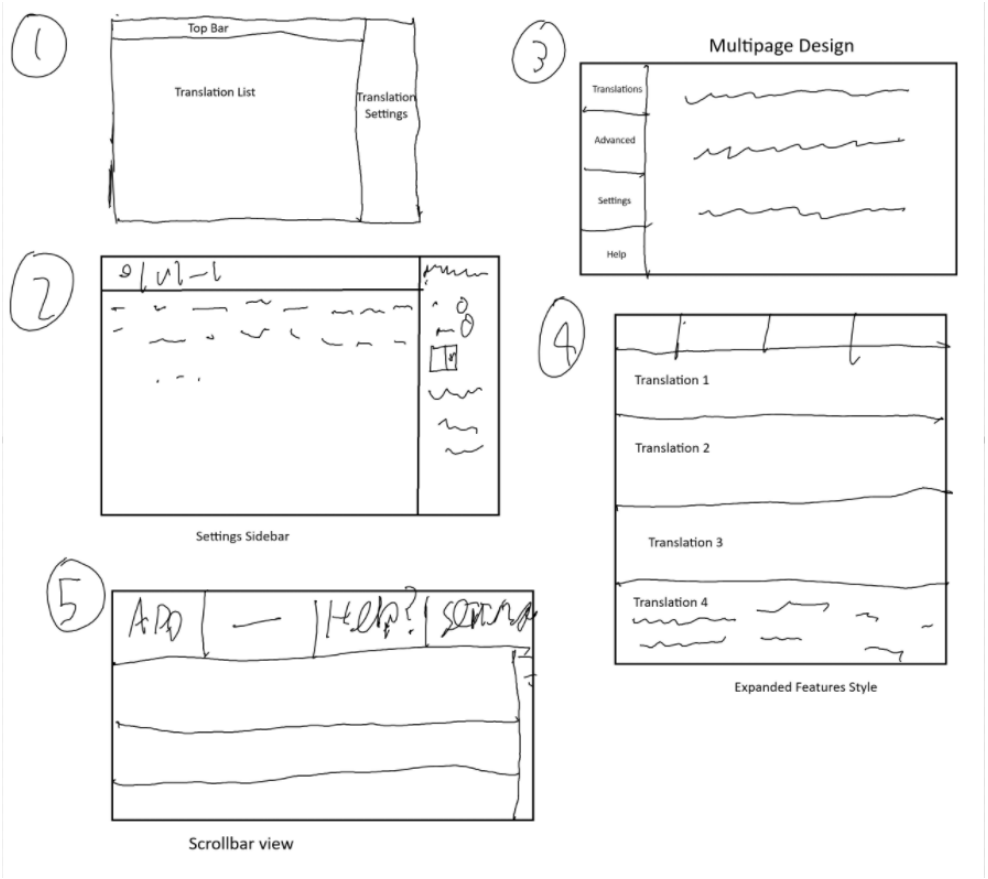
As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

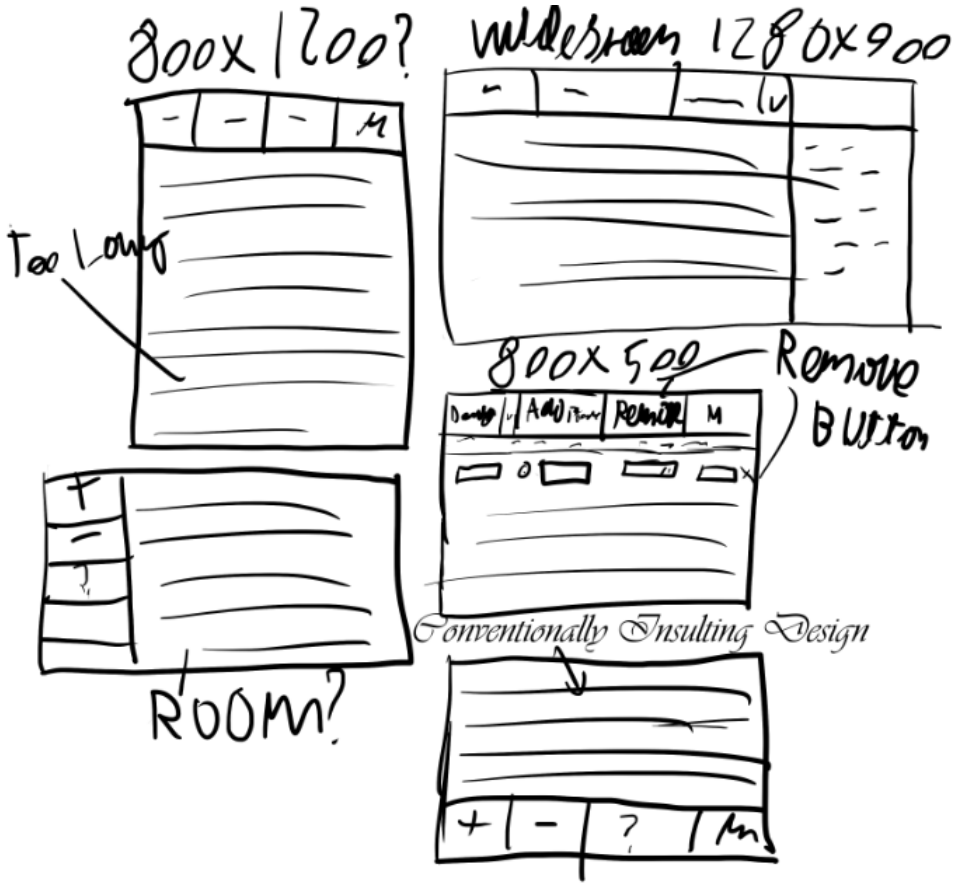
Supervisor comments:

Supervisor's Digital Signature**:** **alexander.gegov@port.ac.uk** Date**: 06/03/2024**

Appendix C

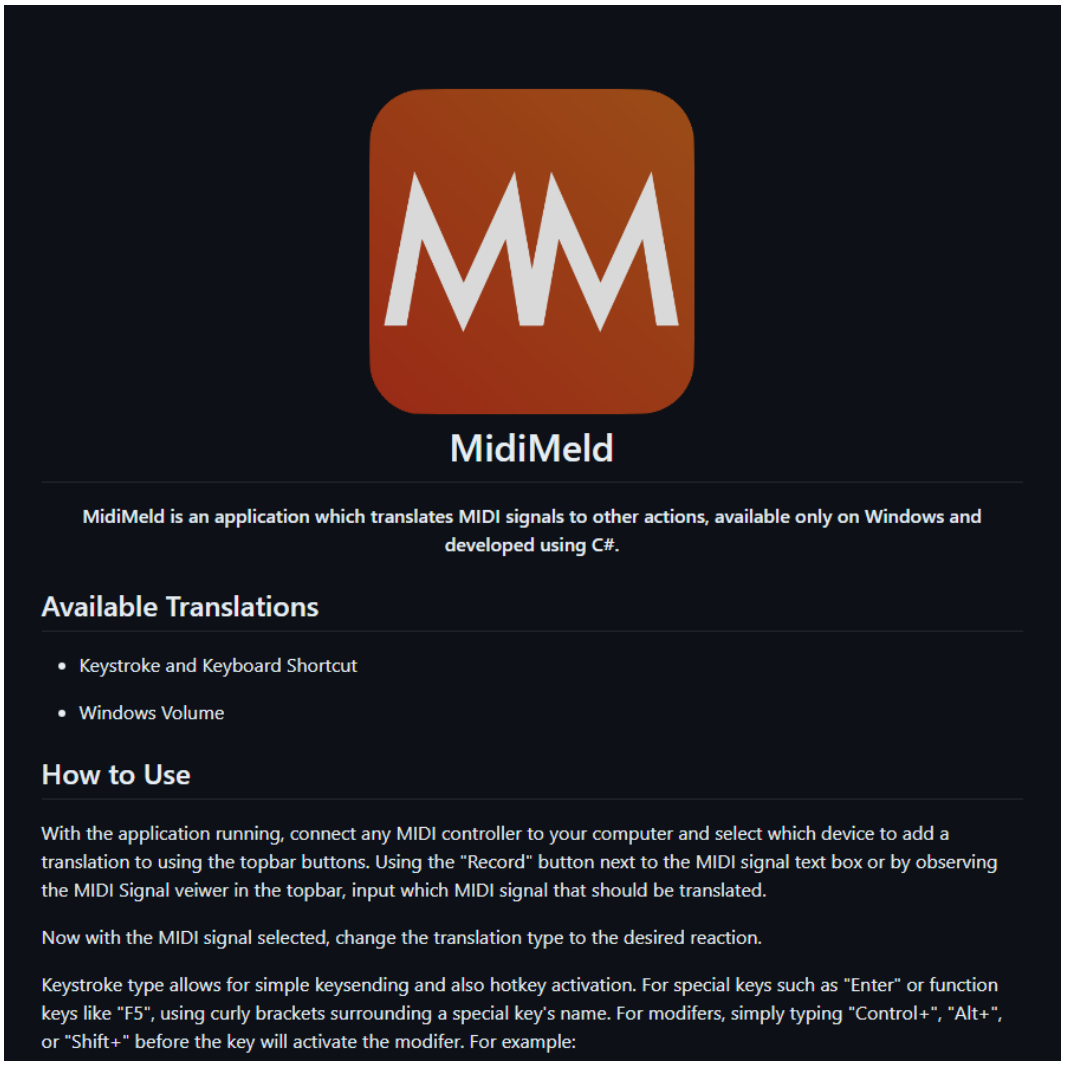
User Interface Design Iterations




800x1200?
Too Long
Widescreen 1280x900
800x500
Remove Button
Conventionally Insulting Design
ROOM?

Appendix D

User Guide Document



MidiMeld

MidiMeld is an application which translates MIDI signals to other actions, available only on Windows and developed using C#.

Available Translations

- Keystroke and Keyboard Shortcut
- Windows Volume

How to Use

With the application running, connect any MIDI controller to your computer and select which device to add a translation to using the topbar buttons. Using the "Record" button next to the MIDI signal text box or by observing the MIDI Signal veiwer in the topbar, input which MIDI signal that should be translated.

Now with the MIDI signal selected, change the translation type to the desired reaction.

Keystroke type allows for simple keysending and also hotkey activation. For special keys such as "Enter" or function keys like "F5", using curly brackets surrounding a special key's name. For modifers, simply typing "Control+", "Alt+", or "Shift+" before the key will activate the modifer. For example:

```
Alt+{F4}
Control+{Enter}
Shift+e
```

Windows Parameter type only currently contains "Volume" which controls allows the control over the default Windows audio device output volume. Primarily used for knobs and dials, this controls volume based on the velocity of the MIDI signal.

Once a translation is set up, when the application detects the desired MIDI signal, it will activate.

Installation

Currently, to install this application for use, simply download the .zip folder from the "Releases" section of this Github page. Unzip the folder and run the "MidiMeld.exe" file.

This application **only runs on Windows**. Whilst this application was developed and tested on Windows 11, it should work from Windows 7+.

Future Work

Some additional features that should be added to this project are:

- Script Execution
- Run in background
- Velocity specific reactions

Development

This application was developed in C# using Microsoft's WPF for UI. The application was made as an artefact for my dissertation for my Bachelors Of Science degree in Software Engineering. This project is still in early development but aims to rival premium and priced applications in the MIDI translation market.

Acknowledgments

Without "DryWetMIDI" by "Melanchall", Mike Meinz, and Hamish Chapman, this project would have never been possible. Thank you dearly.

Appendix E

Study Ethics Documents

CONSENT FORM

Title of Project: Extending Computer Interactivity Using Digital Musical Devices

Name and Contact Details of Researcher(s): Ethan Egerton / up2067147@myport.ac.uk

Name and Contact Details of Supervisor (if relevant): Alexander Gegov / alexander.gegov@port.ac.uk

University Data Protection Officer: Samantha Hill, 023 9284 3642 or information-matters@port.ac.uk

Please tick boxes

1. I confirm that I have read and understood the information sheet. I have had the opportunity to consider the information, ask questions and have had these answered satisfactorily. ☐

2. I understand that my participation is voluntary and that I am free to withdraw at any time during the study without giving any reason. ☐

3. I understand that data collected during this study will be processed in accordance with data protection law as explained in the Participant Information Sheet. ☐

4. I agree to take part in the above study. ☐

Name of Participant: **Date:** **Signature:**

Name of Researcher: Ethan Egerton **Date:** **Signature:**

***Note**: When completed, one copy to be given to the participant, one copy to be retained in the study file*

PARTICIPANT INFORMATION SHEET

Name and Contact Details of Researcher(s): Ethan Egerton / up2067147@myport.ac.uk

Name and Contact Details of Supervisor (if relevant): Alexander Gegov / alexander.gegov@port.ac.uk

1. **Invitation**

   I am Ethan Egerton and am studying Software Engineering at University of Portsmouth.

   I would like to invite you to take part in our research study.

   Joining the study is entirely up to you, before you decide we would like you to understand why the research is being done and what it would involve for you.

   Taking part will take about 10 minutes.

   Please ask/contact us if you have any questions.

2. **Study Summary**

   This purpose of this study is to identify any flaws with the design of an application.

   You will be asked to:

   1. Think aloud whilst using an application
   2. Answer two questions about your experience with application

3. **What data will be collected and / or measurements taken?**

   The data collected from this study will simply be observations about participant experience when using the application. Participant data such as name or age, will **not** be collected as it is irrelevant to the study. All data collected from this study will be destroyed at the end of the project on the 3rd of May.

4. **Do I have to take part?**

   No, taking part in this research is entirely voluntary.

5. **Expenses and payments**

   There is no payment for taking part

6. **What if there is a problem?**

   If you have a query, concern or complaint about any aspect of this study, in the first instance you should contact the researcher(s) if appropriate. Please contact the supervisor listed if you have a complaint.

   Thank you

   Thank you for taking time to read this information sheet and for considering volunteering for this research.

Appendix F

Final Project Code

DeviceHandler.cs

```
<Project Sdk="Microsoft.NET.Sdk.WindowsDesktop">
  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net8.0-windows7.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <UseWPF>true</UseWPF>
    <ApplicationIcon>midimeld.ico</ApplicationIcon>
  </PropertyGroup>

  <ItemGroup>
    <Content Include="midimeld.ico" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Melanchall.DryWetMidi" Version
    ="7.0.2" />
    <Reference Include="WindowsBase" />
    <FrameworkReference Include="Microsoft.WindowsDesktop.App.
    WindowsForms" />
  </ItemGroup>
</Project>
```

DeviceHandler.cs

```
using Melanchall.DryWetMidi.Multimedia;
using Melanchall.DryWetMidi.Tools;
using System;
using System.Diagnostics;
using System.Reflection;
using System.Windows.Forms;

```

```
 8
 9 namespace midism.classes
10 {
11
12     public class DeviceHandler()
13     {
14         public List<string> GetAllDeviceNames()
15         {
16             List<string> deviceNames = [];
17
18             foreach (var inputDevice in InputDevice.GetAll())
19             {
20                 deviceNames.Add(inputDevice.Name);
21             }
22
23             return deviceNames;
24         }
25
26
27         // Check and add perm threaded listener if not listening
28         //
29         public async void AddListenerAndEvent(string deviceName)
30         {
31             // Add perm listener
32             if (!GlobalVariables.activeListeners.Contains(
    deviceName))
33             {
34                 GlobalVariables.activeListeners.Add(deviceName);
35                 await AddPermListener(deviceName);
36             }
37         }
38
39         async Task AddPermListener(string deviceName)
40         {
41             var inputDevice = InputDevice.GetByName(deviceName);
42             while (true)
43             {
44                 if (inputDevice.IsListeningForEvents != true)
45                 {
46                     inputDevice.StartEventsListening();
47                     inputDevice.EventReceived += OnEventReceived;
48                 }
49                 await Task.Delay(500);
50             }
51         }
52
```

```
53          private void OnEventReceived(object sender,
    MidiEventReceivedEventArgs e)
54          {
55              var midiDevice = (MidiDevice)sender;
56              List<int> noteData = NoteParser(e.Event.ToString());
57              int noteNumber = noteData[0];
58              int activation = 0;
59
60              try
61              {
62                  activation = noteData[1];
63              } catch (Exception ex) { }
64
65
66
67              GlobalVariables.MIDIsignal = noteNumber;
68              GlobalVariables.MIDIactivation = activation;
69
70              foreach (KeyValuePair<int, List<String>> pair in
    GlobalVariables.translations)
71              {
72                  System.Diagnostics.Debug.WriteLine(string.Join(",
    ", pair.Value));
73                  string name = pair.Value[0];
74                  int MIDIsignal = int.Parse(pair.Value[1]);
75                  string type = pair.Value[2];
76                  string value = pair.Value[3];
77
78                  if (name != midiDevice.Name ||
79                      MIDIsignal != noteNumber ||
80                      MIDIsignal.GetType() != typeof(int) ||
81                      MIDIsignal == 0 ||
82                      type == "None")
83                  {
84                      continue;
85                  }
86
87                  if (type == "Keyboard Stroke")
88                  {
89                      bool modifierFree = false;
90
91                      if (activation == 0)
92                      {
93                          break;
94                      }
95
96                      while (modifierFree == false)
```

```
97                      {
98                          if (value.Contains("ALT+"))
99                          {
100                             value = value.Remove(value.IndexOf("ALT
    +"), "ALT+".Length);
101                             value = "%" + value;
102                             continue;
103                         }
104                         if (value.Contains("CONTROL+"))
105                         {
106                             value = value.Remove(value.IndexOf("
    CONTROL+"), "CONTROL+".Length);
107                             value = "^" + value;
108                             continue;
109                         }
110                         if (value.Contains("SHIFT+"))
111                         {
112                             value = value.Remove(value.IndexOf("
    SHIFT+"), "SHIFT+".Length);
113                             value = "+" + value;
114                             continue;
115                         }
116                         modifierFree = true;
117                     }
118
119                     System.Diagnostics.Debug.WriteLine(value);
120                     SendKeys.SendWait(value);
121                 }
122
123                 if (type == "Windows Parameter")
124                 {
125                     if (value == "Volume")
126                     {
127                         double volume = ((double)activation / 127)
    * 100;
128                         System.Diagnostics.Debug.WriteLine(
    activation);
129                         System.Diagnostics.Debug.WriteLine(volume);
130                         SystemAudio.WindowsSystemAudio.SetVolume((
    int)volume);
131                     }
132                 }
133             }
134         }
135
136         private List<int> NoteParser(string notes)
137         {
```

```
138             List<int> result = new List<int>();
139 
140             string searchString1 = "(";
141             string searchString2 = ",";
142             string searchString3 = ")";
143 
144             int index1 = notes.IndexOf(searchString1);
145             int index2 = notes.IndexOf(searchString2);
146             int index3 = notes.IndexOf(searchString3);
147             if (index1 == -1 || index2 == -1 || index3 == -1 ||
    notes == null) { return [0, 0]; };
148 
149             if (int.TryParse(notes[(index1 + 1)..index2], out int
    number))
150             {
151                 result.Add(number);
152             }
153 
154             if (int.TryParse(notes[(index2 + 2)..index3], out int
    number2))
155             {
156                 result.Add(number2);
157             }
158 
159             return result;
160         }
161     }
162 }
```

GlobalVariables.cs

```
1 namespace midism.classes
2 {
3     public class GlobalVariables
4     {
5         public static List<string> activeListeners = [];
6         public static List<int> MIDIrecordings = [];
7         public static int MIDIsignal = 0;
8         public static int MIDIactivation = 0;
9         public static int translationCount = 0;
10        public static string selectedDevice = "";
11        public static Dictionary<int, List<string>> translations =
    [];
12    }
13 }
```

WindowsSystemAudio.cs

```
1 using System;
2 using System.Runtime.InteropServices;
3
4
5 // https://www.codeproject.com/Tips/5358341/Getting-and-Setting-
      Windows-System-Audio-Volume
6 namespace SystemAudio
7 {
8     internal static class WindowsSystemAudio
9     {
10         [ComImport]
11         [Guid("A95664D2-9614-4F35-A746-DE8DB63617E6"),
12                InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
13         private interface IMMDeviceEnumerator
14         {
15             void Vtbl();
16             int GetDefaultAudioEndpoint(int dataFlow, int role, out
       IMMDevice ppDevice);
17         }
18         private static class MMDeviceEnumeratorFactory
19         {
20             public static IMMDeviceEnumerator CreateInstance()
21             {
22                 return (IMMDeviceEnumerator)Activator.
      CreateInstance
23                 (Type.GetTypeFromCLSID(new Guid
24                 ("BCDE0395-E52F-467C-8E3D-C4579291692E")));
25             }
26         }
27         [Guid("D666063F-1587-4E43-81F1-B948E807363F"),
28                InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
29         private interface IMMDevice
30         {
31             int Activate([MarshalAs(UnmanagedType.LPStruct)] Guid
      iid,
32                          int dwClsCtx, IntPtr pActivationParams,
33                          [MarshalAs(UnmanagedType.IUnknown)] out
      object ppInterface);
34         }
35
36         [Guid("5CDF2C82-841E-4546-9722-0CF74078229A"),
37                InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
38         public interface IAudioEndpointVolume
39         {
40             int RegisterControlChangeNotify(IntPtr pNotify);
41             int UnregisterControlChangeNotify(IntPtr pNotify);
42             int GetChannelCount(ref uint pnChannelCount);
```

```
43              int SetMasterVolumeLevel(float fLevelDB, Guid
    pguidEventContext);
44              int SetMasterVolumeLevelScalar(float fLevel, Guid
    pguidEventContext);
45              int GetMasterVolumeLevel(ref float pfLevelDB);
46              int GetMasterVolumeLevelScalar(ref float pfLevel);
47          }
48
49          internal static void SetVolume(int level)
50          {
51              try
52              {
53                  IMMDeviceEnumerator deviceEnumerator =
54                                      MMDeviceEnumeratorFactory.
    CreateInstance();
55                  IMMDevice speakers;
56                  const int eRender = 0;
57                  const int eMultimedia = 1;
58                  deviceEnumerator.GetDefaultAudioEndpoint
59                                      (eRender, eMultimedia, out
    speakers);
60                  object aepv_obj;
61                  speakers.Activate(typeof(IAudioEndpointVolume).GUID
    ,
62                                       0, IntPtr.Zero, out aepv_obj);
63                  IAudioEndpointVolume aepv = (IAudioEndpointVolume)
    aepv_obj;
64                  Guid ZeroGuid = new();
65                  int res = aepv.SetMasterVolumeLevelScalar(level /
    100f, ZeroGuid);
66              }
67              catch (Exception ex)
68              {
69              }
70          }
71      }
72 }
```

App.xaml

```
1 <Application x:Class="midism.App"
2              xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
    presentation"
3              xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4              xmlns:local="clr-namespace:midism"
5              StartupUri="MainWindow.xaml">
6     <Application.Resources>
```

```
7         <BitmapImage x:Key="AppIcon" UriSource="assets/midimeld.ico
    "/>
8     </Application.Resources>
9 </Application>
```

App.xaml.cs

```
1 using System.Windows;
2
3 namespace midism
4 {
5     /// <summary>
6     /// Interaction logic for App.xaml
7     /// </summary>
8     public partial class App : Application
9     {
10    }
11 }
```

AssemblyInfo.cs

```
1 using System.Windows;
2
3 [assembly: ThemeInfo(
4     ResourceDictionaryLocation.None,            //where theme
    specific resource dictionaries are located
5                                                 //(used if a
    resource is not found in the page,
6                                                 // or application
    resource dictionaries)
7     ResourceDictionaryLocation.SourceAssembly   //where the generic
     resource dictionary is located
8                                                 //(used if a
    resource is not found in the page,
9                                                 // app, or any
    theme specific resource dictionaries)
10 )]
```

MainWindow.xaml

```
1 <NavigationWindow x:Class="midism.MainWindow"
2         xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
    presentation"
3         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4         xmlns:d="http://schemas.microsoft.com/expression/blend
    /2008"
5         xmlns:mc="http://schemas.openxmlformats.org/markup-
    compatibility/2006"
```

```
6         xmlns:local="clr-namespace:midism"
7         mc:Ignorable="d"
8         Title="MidiMeld" Height="450" Width="800" Source="UI.xaml"
      ResizeMode="NoResize"
9         WindowStartupLocation="CenterScreen" ShowsNavigationUI="
      False">
10 </NavigationWindow>
```

MainWindow.xaml.cs

```
1 using System.Windows.Navigation;
2
3 namespace midism
4 {
5
6     /// <summary>
7     /// Interaction logic for MainWindow.xaml
8     /// </summary>
9     public partial class MainWindow : NavigationWindow
10    {
11        public MainWindow()
12        {
13            InitializeComponent();
14        }
15    }
16 }
```

UI.xaml

```
1 <Page x:Class="midism.UI"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
      presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       xmlns:mc="http://schemas.openxmlformats.org/markup-
      compatibility/2006"
5       xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
6       xmlns:local="clr-namespace:midism"
7       mc:Ignorable="d"
8       d:DesignHeight="450" d:DesignWidth="800"
9       Title="UI" ShowsNavigationUI="False"
10      FontFamily="Century Gothic"
11      Foreground="White"
12      Background="#231B1D"
13      Loaded="Page_Loaded">
14
15
16
17    <Page.Resources>
```

```
18          <Style TargetType="{x:Type Button}">
19              <Setter Property="Background" Value="#231B1D"/>
20              <Setter Property="Template">
21                  <Setter.Value>
22                      <ControlTemplate TargetType="{x:Type Button}">
23                          <Border Background="{TemplateBinding
    Background}" BorderBrush="#B07F6D" BorderThickness="1">
24                              <ContentPresenter HorizontalAlignment="
    Center" VerticalAlignment="Center"/>
25                          </Border>
26                      </ControlTemplate>
27                  </Setter.Value>
28              </Setter>
29              <Style.Triggers>
30                  <Trigger Property="IsMouseOver" Value="True">
31                      <Setter Property="Background" Value="#47232C"/>
32                  </Trigger>
33              </Style.Triggers>
34          </Style>
35
36
37          <!-- https://learn.microsoft.com/en-us/previous-versions/
    dotnet/netframework-3.0/ms752094(v=vs.85) -->
38
39          <LinearGradientBrush x:Key="NormalBrush" StartPoint="0,0"
    EndPoint="0,1">
40              <GradientBrush.GradientStops>
41                  <GradientStopCollection>
42                      <GradientStop Color="#231B1D" Offset="0.0"/>
43                      <GradientStop Color="#231B1D" Offset="1.0"/>
44                  </GradientStopCollection>
45              </GradientBrush.GradientStops>
46          </LinearGradientBrush>
47
48          <LinearGradientBrush x:Key="HorizontalNormalBrush"
    StartPoint="0,0" EndPoint="1,0">
49              <GradientBrush.GradientStops>
50                  <GradientStopCollection>
51                      <GradientStop Color="#FFF" Offset="0.0"/>
52                      <GradientStop Color="#CCC" Offset="1.0"/>
53                  </GradientStopCollection>
54              </GradientBrush.GradientStops>
55          </LinearGradientBrush>
56
57          <LinearGradientBrush x:Key="LightBrush" StartPoint="0,0"
    EndPoint="0,1">
58              <GradientBrush.GradientStops>
```

```
59                <GradientStopCollection>
60                    <GradientStop Color="#FFF" Offset="0.0"/>
61                    <GradientStop Color="#EEE" Offset="1.0"/>
62                </GradientStopCollection>
63            </GradientBrush.GradientStops>
64        </LinearGradientBrush>
65
66        <LinearGradientBrush x:Key="HorizontalLightBrush"
    StartPoint="0,0" EndPoint="1,0">
67            <GradientBrush.GradientStops>
68                <GradientStopCollection>
69                    <GradientStop Color="#FFF" Offset="0.0"/>
70                    <GradientStop Color="#EEE" Offset="1.0"/>
71                </GradientStopCollection>
72            </GradientBrush.GradientStops>
73        </LinearGradientBrush>
74
75        <LinearGradientBrush x:Key="DarkBrush" StartPoint="0,0"
    EndPoint="0,1">
76            <GradientBrush.GradientStops>
77                <GradientStopCollection>
78                    <GradientStop Color="#47232C" Offset="0.0"/>
79                    <GradientStop Color="#47232C" Offset="1.0"/>
80                </GradientStopCollection>
81            </GradientBrush.GradientStops>
82        </LinearGradientBrush>
83
84        <LinearGradientBrush x:Key="PressedBrush" StartPoint="0,0"
    EndPoint="0,1">
85            <GradientBrush.GradientStops>
86                <GradientStopCollection>
87                    <GradientStop Color="#231B1D" Offset="0.0"/>
88                    <GradientStop Color="#231B1D" Offset="0.1"/>
89                    <GradientStop Color="#231B1D" Offset="0.9"/>
90                    <GradientStop Color="#231B1D" Offset="1.0"/>
91                </GradientStopCollection>
92            </GradientBrush.GradientStops>
93        </LinearGradientBrush>
94
95        <SolidColorBrush x:Key="DisabledForegroundBrush" Color
    ="#231B1D" />
96
97        <SolidColorBrush x:Key="DisabledBackgroundBrush" Color
    ="#231B1D" />
98
99        <SolidColorBrush x:Key="WindowBackgroundBrush" Color="#231
    B1D" />
```

```
100
101         <SolidColorBrush x:Key="SelectedBackgroundBrush" Color
        ="#231B1D" />
102
103         <!-- Border Brushes -->
104
105         <LinearGradientBrush x:Key="NormalBorderBrush" StartPoint
        ="0,0" EndPoint="0,1">
106             <GradientBrush.GradientStops>
107                 <GradientStopCollection>
108                     <GradientStop Color="#B07F6D" Offset="0.0"/>
109                     <GradientStop Color="#B07F6D" Offset="1.0"/>
110                 </GradientStopCollection>
111             </GradientBrush.GradientStops>
112         </LinearGradientBrush>
113
114         <LinearGradientBrush x:Key="HorizontalNormalBorderBrush"
        StartPoint="0,0" EndPoint="1,0">
115             <GradientBrush.GradientStops>
116                 <GradientStopCollection>
117                     <GradientStop Color="#B07F6D" Offset="0.0"/>
118                     <GradientStop Color="#B07F6D" Offset="1.0"/>
119                 </GradientStopCollection>
120             </GradientBrush.GradientStops>
121         </LinearGradientBrush>
122
123         <LinearGradientBrush x:Key="DefaultedBorderBrush"
        StartPoint="0,0" EndPoint="0,1">
124             <GradientBrush.GradientStops>
125                 <GradientStopCollection>
126                     <GradientStop Color="#B07F6D" Offset="0.0"/>
127                     <GradientStop Color="#B07F6D" Offset="1.0"/>
128                 </GradientStopCollection>
129             </GradientBrush.GradientStops>
130         </LinearGradientBrush>
131
132         <LinearGradientBrush x:Key="PressedBorderBrush" StartPoint
        ="0,0" EndPoint="0,1">
133             <GradientBrush.GradientStops>
134                 <GradientStopCollection>
135                     <GradientStop Color="#B07F6D" Offset="0.0"/>
136                     <GradientStop Color="#B07F6D" Offset="1.0"/>
137                 </GradientStopCollection>
138             </GradientBrush.GradientStops>
139         </LinearGradientBrush>
140
```

```
141         <SolidColorBrush x:Key="DisabledBorderBrush" Color="#B07F6D
    "/>
142 
143         <SolidColorBrush x:Key="SolidBorderBrush" Color="#B07F6D"/>
144 
145         <SolidColorBrush x:Key="LightBorderBrush" Color="#B07F6D"/>
146 
147         <!-- Miscellaneous Brushes -->
148         <SolidColorBrush x:Key="GlyphBrush" Color="#DDD"/>
149 
150         <SolidColorBrush x:Key="LightColorBrush" Color="#DDD"/>
151 
152         <ControlTemplate x:Key="ComboBoxToggleButton" TargetType="
    ToggleButton">
153             <Grid>
154                 <Grid.ColumnDefinitions>
155                     <ColumnDefinition />
156                     <ColumnDefinition Width="26"/>
157                 </Grid.ColumnDefinitions>
158                 <Border
159                   x:Name="Border"
160                   Grid.ColumnSpan="2"
161                   CornerRadius="2"
162                   Background="{StaticResource NormalBrush}"
163                   BorderBrush="{StaticResource NormalBorderBrush}"
164                   BorderThickness="1" />
165                 <Border
166                   Grid.Column="0"
167                   CornerRadius="2,0,0,2"
168                   Margin="1"
169                   Background="{StaticResource WindowBackgroundBrush
    }"
170                   BorderBrush="{StaticResource NormalBorderBrush}"
171                   BorderThickness="0,0,1,0" />
172 
173                 <Path
174                               x:Name="Arrow"
175                               Grid.Column="1"
176                               Fill="{StaticResource GlyphBrush}"
177                               Margin="0 0 3 0"
178                               HorizontalAlignment="Center"
179                               VerticalAlignment="Center"
180                               Data="M 0 0 L 4 4 L 8 0 Z"/>
181             </Grid>
182             <ControlTemplate.Triggers>
183                 <Trigger Property="ToggleButton.IsMouseOver" Value
    ="true">
```

```
184                     <Setter TargetName="Border" Property="
    Background" Value="{StaticResource DarkBrush}" />
185                 </Trigger>
186                 <Trigger Property="ToggleButton.IsChecked" Value="
    true">
187                     <Setter TargetName="Border" Property="
    Background" Value="{StaticResource PressedBrush}" />
188                 </Trigger>
189                 <Trigger Property="IsEnabled" Value="False">
190                     <Setter TargetName="Border" Property="
    Background" Value="{StaticResource DisabledBackgroundBrush}" />
191                     <Setter TargetName="Border" Property="
    BorderBrush" Value="{StaticResource DisabledBorderBrush}" />
192                     <Setter Property="Foreground" Value="{
    StaticResource DisabledForegroundBrush}"/>
193                     <Setter TargetName="Arrow" Property="Fill"
    Value="{StaticResource DisabledForegroundBrush}" />
194                 </Trigger>
195             </ControlTemplate.Triggers>
196         </ControlTemplate>
197 
198         <ControlTemplate x:Key="ComboBoxTextBox" TargetType="
    TextBox">
199             <Border x:Name="PART_ContentHost" Focusable="False"
    Background="{TemplateBinding Background}" />
200         </ControlTemplate>
201 
202         <Style x:Key="{x:Type ComboBox}" TargetType="ComboBox">
203             <Setter Property="SnapsToDevicePixels" Value="true"/>
204             <Setter Property="OverridesDefaultStyle" Value="true"/>
205             <Setter Property="ScrollViewer.
    HorizontalScrollBarVisibility" Value="Auto"/>
206             <Setter Property="ScrollViewer.
    VerticalScrollBarVisibility" Value="Auto"/>
207             <Setter Property="ScrollViewer.CanContentScroll" Value
    ="true"/>
208             <Setter Property="MinWidth" Value="120"/>
209             <Setter Property="MinHeight" Value="20"/>
210             <Setter Property="Template">
211                 <Setter.Value>
212                     <ControlTemplate TargetType="ComboBox">
213                         <Grid>
214                             <ToggleButton
215                                 Name="ToggleButton"
216                                 Template="{StaticResource
    ComboBoxToggleButton}"
217                                 Grid.Column="2"
```

```
218                                 Focusable="false"
219                                 IsChecked="{Binding Path=
    IsDropDownOpen,Mode=TwoWay,RelativeSource={RelativeSource
    TemplatedParent}}"
220                                 ClickMode="Press">
221                             </ToggleButton>
222                             <ContentPresenter
223                                 Name="ContentSite"
224                                 IsHitTestVisible="False"
225                                 Content="{TemplateBinding
    SelectionBoxItem}"
226                                 ContentTemplate="{TemplateBinding
    SelectionBoxItemTemplate}"
227                                 ContentTemplateSelector="{
    TemplateBinding ItemTemplateSelector}"
228                                 Margin="3,3,23,3"
229                                 VerticalAlignment="Center"
230                                 HorizontalAlignment="Center" />
231                             <TextBox x:Name="PART_EditableTextBox"
232                                 Style="{x:Null}"
233                                 Template="{StaticResource
    ComboBoxTextBox}"
234                                 HorizontalAlignment="Center"
235                                 VerticalAlignment="Center"
236                                 Margin="3,3,23,3"
237                                 Focusable="True"
238                                 Background="Transparent"
239                                 Visibility="Hidden"
240                                 IsReadOnly="{TemplateBinding
    IsReadOnly}"/>
241                             <Popup
242                                 Name="Popup"
243                                 Placement="Bottom"
244                                 IsOpen="{TemplateBinding
    IsDropDownOpen}"
245                                 AllowsTransparency="True"
246                                 Focusable="False"
247                                 PopupAnimation="Slide">
248                                 <Grid
249                                   Name="DropDown"
250                                   SnapsToDevicePixels="True"
251                                   MinWidth="{TemplateBinding
    ActualWidth}"
252                                   MaxHeight="{TemplateBinding
    MaxDropDownHeight}">
253                                     <Border
254                                         x:Name="DropDownBorder"
```

```
255                                     Background="{StaticResource
     WindowBackgroundBrush}"
256                                     BorderThickness="1"
257                                     BorderBrush="{
    StaticResource SolidBorderBrush}"/>
258                                 <ScrollViewer Margin="4,6,4,6"
    SnapsToDevicePixels="True">
259                                     <StackPanel IsItemsHost="
    True" KeyboardNavigation.DirectionalNavigation="Contained" />
260                                 </ScrollViewer>
261                             </Grid>
262                         </Popup>
263                     </Grid>
264                     <ControlTemplate.Triggers>
265                         <Trigger Property="HasItems" Value="
    false">
266                             <Setter TargetName="DropDownBorder"
     Property="MinHeight" Value="95"/>
267                         </Trigger>
268                         <Trigger Property="IsEnabled" Value="
    false">
269                             <Setter Property="Foreground" Value
    ="{StaticResource DisabledForegroundBrush}"/>
270                         </Trigger>
271                         <Trigger Property="IsGrouping" Value="
    true">
272                             <Setter Property="ScrollViewer.
    CanContentScroll" Value="false"/>
273                         </Trigger>
274                         <Trigger SourceName="Popup" Property="
    Popup.AllowsTransparency" Value="true">
275                             <Setter TargetName="DropDownBorder"
     Property="CornerRadius" Value="4"/>
276                             <Setter TargetName="DropDownBorder"
     Property="Margin" Value="0,2,0,0"/>
277                         </Trigger>
278                         <Trigger Property="IsEditable" Value="
    true">
279                             <Setter Property="IsTabStop" Value
    ="false"/>
280                             <Setter TargetName="
    PART_EditableTextBox" Property="Visibility"    Value="Visible"/>
281                             <Setter TargetName="ContentSite"
    Property="Visibility" Value="Hidden"/>
282                         </Trigger>
283                     </ControlTemplate.Triggers>
284                 </ControlTemplate>
```

```
285                 </Setter.Value>
286             </Setter>
287             <Style.Triggers>
288             </Style.Triggers>
289         </Style>
290     </Page.Resources>
291
292
293     <Grid Name="MainGrid">
294         <Grid.RowDefinitions>
295             <RowDefinition Height="80"/>
296             <RowDefinition/>
297         </Grid.RowDefinitions>
298
299         <Grid.ColumnDefinitions>
300             <ColumnDefinition/>
301             <ColumnDefinition/>
302             <ColumnDefinition/>
303             <ColumnDefinition/>
304         </Grid.ColumnDefinitions>
305
306         <StackPanel
307             Grid.Row="0"
308             Grid.Column="0"
309             VerticalAlignment="Center"
310             HorizontalAlignment="Center"
311             Background="#231B1D">
312             <Button Height="80" Click="Button_Click" Width="198"
    BorderThickness="0" FontSize="20" Foreground="White">
313                 Add Translation
314             </Button>
315
316         </StackPanel>
317
318         <StackPanel
319             Grid.Row="0"
320             Grid.Column="1"
321             Background="White"
322             VerticalAlignment="Center"
323             HorizontalAlignment="Center">
324             <ComboBox Name="Combobox1"
325                       Height="80"
326                       DropDownOpened="Combobox1_DropDownOpened"
    Loaded="Combobox1_Loaded" SelectionChanged="
    Combobox1_SelectionChanged"
```

```
327                     VerticalContentAlignment="Center"
    HorizontalContentAlignment="Center" VerticalAlignment="Center"
    HorizontalAlignment="Center"
328                     SelectedIndex="0" Width="198" BorderThickness
    ="0,0,0,1" BorderBrush="#B07F6D" FontSize="20">
329                 <ComboBox.ItemTemplate>
330                     <DataTemplate>
331                         <TextBlock Text="{Binding}"
    VerticalAlignment="Center" HorizontalAlignment="Center"/>
332                     </DataTemplate>
333                 </ComboBox.ItemTemplate>
334             </ComboBox>
335         </StackPanel>
336 
337         <StackPanel
338             Grid.Row="0"
339             Grid.Column="2"
340             Background="Transparent"
341             Margin="7 5 0 0">
342             <Grid Height="80">
343                 <Grid.ColumnDefinitions>
344                     <ColumnDefinition/>
345                     <ColumnDefinition/>
346                 </Grid.ColumnDefinitions>
347 
348                 <TextBlock Height="30" Background="Transparent"
    TextAlignment="Center" Width="96" FontSize="18" Grid.Column="0"
    VerticalAlignment="Top">
349                     MIDI Signal
350                 </TextBlock>
351                 <TextBlock Name="MIDISignalBox" Height="50"
    Background="Transparent" TextAlignment="Center" Width="96"
    FontSize="30" Grid.Column="0" VerticalAlignment="Bottom">
352                      None
353                 </TextBlock>
354                 <TextBlock Height="30" Background="Transparent"
    TextAlignment="Center" Width="96" FontSize="18" Grid.Column="1"
    VerticalAlignment="Top">
355                     Velocity
356                 </TextBlock>
357                 <TextBlock Name="VelocityBox" Height="50"
    Background="Transparent" TextAlignment="Center" Width="96"
    FontSize="30" Grid.Column="1" VerticalAlignment="Top" Margin
    ="2,30,2,0">
358                     None
359                 </TextBlock>
360             </Grid>
```

```
361         </StackPanel>
362 
363         <Border Grid.Column="0" Grid.Row="0" BorderBrush="#B07F6D"
    BorderThickness="0 0 1 2" />
364         <Border Grid.Column="1" Grid.Row="0" BorderBrush="#B07F6D"
    BorderThickness="0 0 3 2" />
365         <Border Grid.Column="2" Grid.Row="0" BorderBrush="#B07F6D"
    BorderThickness="0 0 1 2" />
366         <Border Grid.Column="3" Grid.Row="0" BorderBrush="#B07F6D"
    BorderThickness="0 0 0 2" />
367 
368         <ScrollViewer Grid.Row="1" Grid.ColumnSpan="4" Padding="5"
369                       VerticalScrollBarVisibility="Auto">
370             <Grid
371             Name="TranslationGrid">
372                 <Grid.RowDefinitions>
373                     <RowDefinition Height="22"/>
374                 </Grid.RowDefinitions>
375                 <Grid.ColumnDefinitions>
376                     <ColumnDefinition Width="40"/>
377                     <ColumnDefinition Width="160"/>
378                     <ColumnDefinition Width="40"/>
379                     <ColumnDefinition Width="160"/>
380                     <ColumnDefinition Width="200"/>
381                     <ColumnDefinition Width="200"/>
382                 </Grid.ColumnDefinitions>
383                 <TextBlock Grid.Row="0" Grid.Column="0" FontSize
    ="14" TextAlignment="Center">
384                     Index
385                 </TextBlock>
386                 <TextBlock Grid.Row="0" Grid.Column="1" FontSize
    ="14" TextAlignment="Center">
387                     Device
388                 </TextBlock>
389                 <TextBlock Grid.Row="0" Grid.Column="2" FontSize
    ="14" TextAlignment="Center">
390                     <Run Text="UTF Black Circle" Foreground="Red"/>
391                     <Run Text="?"/>
392                 </TextBlock>
393                 <TextBlock Grid.Row="0" Grid.Column="3" FontSize
    ="14" TextAlignment="Left" Margin="15 0 0 0">
394                     MIDI Signal
395                 </TextBlock>
396                 <TextBlock Grid.Row="0" Grid.Column="4" FontSize
    ="14" TextAlignment="Left" Margin="15 0 0 0">
397                     Translation Type
398                 </TextBlock>
```

```
399                 <TextBlock Grid.Row="0" Grid.Column="5" FontSize
    ="14" TextAlignment="Left" Margin="20 0 0 0">
400                     Translation
401                 </TextBlock>
402                 <Border Grid.ColumnSpan="6" Grid.Row="0"
    BorderBrush="#B07F6D" BorderThickness="0 0 0 0.5"/>
403
404             </Grid>
405         </ScrollViewer>
406     </Grid>
407 </Page>
```

UI.xaml.cs

```
 1 using midism.classes;
 2 using System.Collections.Generic;
 3 using System.Security.Cryptography.X509Certificates;
 4 using System.Windows;
 5 using System.Windows.Controls;
 6 using System.Windows.Input;
 7 using System.Windows.Media;
 8 using System.Xml.Linq;
 9
10 namespace midism
11 {
12     /// <summary>
13     /// Interaction logic for UI.xaml
14     /// </summary>
15     public partial class UI : Page
16     {
17         public UI()
18         {
19             InitializeComponent();
20         }
21
22         private bool loaded = false;
23
24         private async void Page_Loaded(object sender,
    RoutedEventArgs e)
25         {
26             await MIDISignalTextBoxListener();
27         }
28
29         private async Task MIDISignalTextBoxListener()
30         {
31             int signalCache = GlobalVariables.MIDIsignal;
32             int velocityCache = GlobalVariables.MIDIactivation;
```

```
33
34              while (true)
35              {
36                  if (signalCache != GlobalVariables.MIDIsignal ||
    velocityCache != GlobalVariables.MIDIactivation)
37                  {
38                      signalCache = GlobalVariables.MIDIsignal;
39                      MIDISignalBox.Text = signalCache.ToString();
40
41                      velocityCache = GlobalVariables.MIDIactivation;
42                      VelocityBox.Text = velocityCache.ToString();
43                  }
44
45                  if (GlobalVariables.MIDIrecordings.Count != 0)
46                  {
47                      for (int i = 0; i < GlobalVariables.
    MIDIrecordings.Count; i++)
48                      {
49                          int rowIndex = GlobalVariables.
    MIDIrecordings[i];
50                          UIElement element =
    GetChildElementAtPosition(TranslationGrid, rowIndex, 3);
51                          if (element is TextBox textBox)
52                          {
53                              textBox.Text = signalCache.ToString();
54                          }
55                      }
56                  }
57
58                  await Task.Delay(50);
59              }
60          }
61
62
63          // Combobox1
64          private void Combobox1_DropDownOpened(object sender,
    EventArgs e)
65          {
66              var DeviceHandlerClass = new DeviceHandler();
67              var deviceNames = DeviceHandlerClass.GetAllDeviceNames
    ();
68              var comboBoxIndex = Combobox1.SelectedIndex;
69
70              Combobox1.Items.Clear();
71              Combobox1.Items.Add("None");
72
73              foreach (var name in deviceNames)
```

```
74              {
75                  Combobox1.Items.Add(name);
76              }
77
78              if (comboBoxIndex > Combobox1.Items.Count)
79              {
80                  Combobox1.SelectedIndex = 0;
81              }
82              else
83              {
84                  Combobox1.SelectedIndex = comboBoxIndex;
85              }
86          }
87
88          private void Combobox1_Loaded(object sender,
    RoutedEventArgs e)
89          {
90              Combobox1.Items.Add("None");
91          }
92
93          private void Combobox1_SelectionChanged(object sender,
    EventArgs e)
94          {
95              if (Combobox1.SelectedValue != null)
96              {
97                  GlobalVariables.selectedDevice = Combobox1.
    SelectedValue.ToString();
98              }
99
100         }
101
102         // Add Translation Button
103         private void Button_Click(object sender, RoutedEventArgs e)
104         {
105             loaded = false;
106             var DeviceHandlerClass = new DeviceHandler();
107             var deviceName = Combobox1.SelectedValue as string;
108
109             if (deviceName == "None" || deviceName == null)
110             {
111                 return;
112             }
113
114             DeviceHandlerClass.AddListenerAndEvent(deviceName);
115
116             var rowIndex = GlobalVariables.translationCount + 1;
117             GlobalVariables.translationCount += 1;
```

```
118
119             // Row Border
120             Border rowBorder = new Border
121             {
122                 BorderBrush = new SolidColorBrush(Color.FromRgb
    (176, 127, 109)),
123                 BorderThickness = new Thickness(0, 0, 0, 1),
124                 Background = Brushes.Transparent
125             };
126
127             // Add new row
128             RowDefinition newRow = new RowDefinition();
129             newRow.Height = new GridLength(50);
130
131             TranslationGrid.RowDefinitions.Add(newRow);
132             Grid.SetRow(rowBorder, rowIndex);
133             Grid.SetColumnSpan(rowBorder, 6);
134             TranslationGrid.Children.Add(rowBorder);
135
136             // Numarate Row
137             TextBlock rowNumber = new TextBlock();
138             rowNumber.Name = "index" + rowIndex;
139             rowNumber.Text = GlobalVariables.translationCount.
    ToString();
140             rowNumber.FontSize = 12;
141             rowNumber.VerticalAlignment = VerticalAlignment.Center;
142             rowNumber.HorizontalAlignment = HorizontalAlignment.
    Center;
143
144             Grid.SetColumn(rowNumber, 0);
145             Grid.SetRow(rowNumber, rowIndex);
146             TranslationGrid.Children.Add(rowNumber);
147
148
149             // Add listening device combo
150             ComboBox deviceSelector = new ComboBox();
151             deviceSelector.Name = "deviceSelector" + rowIndex;
152             deviceSelector.Loaded += DeviceSelector_Loaded;
153             deviceSelector.DropDownOpened +=
    DeviceSelector_DropDownOpened;
154             deviceSelector.DropDownClosed += setTranslationsCombo;
155             deviceSelector.SelectedIndex = deviceSelector.Items.
    IndexOf(deviceName);
156             deviceSelector.VerticalAlignment = VerticalAlignment.
    Center;
157             deviceSelector.HorizontalAlignment =
    HorizontalAlignment.Center;
```

```
158             deviceSelector.VerticalContentAlignment =
    VerticalAlignment.Center;
159             deviceSelector.HorizontalContentAlignment =
    HorizontalAlignment.Center;
160             deviceSelector.Width = 140;
161             deviceSelector.Height = 25;
162
163             Grid.SetColumn(deviceSelector, 1);
164             Grid.SetRow(deviceSelector, rowIndex);
165             TranslationGrid.Children.Add(deviceSelector);
166
167             // Add MIDI signal box
168
169             CheckBox recordButton = new CheckBox();
170             recordButton.Name = "recordButton" + rowIndex;
171             recordButton.VerticalAlignment = VerticalAlignment.
    Center;
172             recordButton.HorizontalAlignment = HorizontalAlignment.
    Center;
173             recordButton.VerticalContentAlignment =
    VerticalAlignment.Center;
174             recordButton.HorizontalContentAlignment =
    HorizontalAlignment.Center;
175             recordButton.Checked += Checkbox_Changed;
176             recordButton.Unchecked += Checkbox_Changed;
177
178             TextBox MIDIsignalText = new TextBox();
179             MIDIsignalText.Name = "MIDISignal" + rowIndex;
180             MIDIsignalText.PreviewTextInput +=
    TextBox_PreviewTextInput;
181             MIDIsignalText.TextChanged += setTranslationsTextbox;
182             MIDIsignalText.Width = 120;
183             MIDIsignalText.Height = 25;
184             MIDIsignalText.VerticalAlignment = VerticalAlignment.
    Center;
185             MIDIsignalText.HorizontalAlignment =
    HorizontalAlignment.Left;
186             MIDIsignalText.VerticalContentAlignment =
    VerticalAlignment.Center;
187             MIDIsignalText.HorizontalContentAlignment =
    HorizontalAlignment.Left;
188
189             Grid.SetColumn(recordButton, 2);
190             Grid.SetColumn(MIDIsignalText, 3);
191             Grid.SetRow(recordButton, rowIndex);
192             Grid.SetRow(MIDIsignalText, rowIndex);
193             TranslationGrid.Children.Add(recordButton);
```

```
194             TranslationGrid.Children.Add(MIDIsignalText);
195 
196             // Add reaction box
197             ComboBox typeSelector = new ComboBox();
198             typeSelector.Name = "typeSelector" + rowIndex;
199             typeSelector.SelectionChanged += setTranslationsCombo;
200             typeSelector.VerticalAlignment = VerticalAlignment.
    Center;
201             typeSelector.HorizontalAlignment = HorizontalAlignment.
    Left;
202             typeSelector.VerticalContentAlignment =
    VerticalAlignment.Center;
203             typeSelector.HorizontalContentAlignment =
    HorizontalAlignment.Center;
204             typeSelector.Width = 160;
205             typeSelector.Height = 25;
206             typeSelector.Items.Add("None");
207             typeSelector.Items.Add("Keyboard Stroke");
208             typeSelector.Items.Add("Windows Parameter");
209             typeSelector.SelectedIndex = 0;
210             typeSelector.SelectionChanged +=
    typeSelector_SelectionChanged;
211 
212             Grid.SetColumn(typeSelector, 4);
213             Grid.SetRow(typeSelector, rowIndex);
214             TranslationGrid.Children.Add(typeSelector);
215 
216             loaded = true;
217         }
218 
219         // Device Selector Combobox
220         private void DeviceSelector_Loaded(object sender, EventArgs
     e)
221         {
222             if (sender is ComboBox comboBox)
223             {
224                 DeviceSelector_DropDownOpened(sender, e);
225                 comboBox.SelectedIndex = Combobox1.SelectedIndex;
226             }
227         }
228 
229         private void DeviceSelector_DropDownOpened(object sender,
    EventArgs e)
230         {
231             if (sender is ComboBox comboBox)
232             {
233                 var DeviceHandlerClass = new DeviceHandler();
```

```
234                 var deviceNames = DeviceHandlerClass.
    GetAllDeviceNames();
235                 var comboBoxIndex = comboBox.SelectedIndex;
236 
237                 comboBox.Items.Clear();
238                 comboBox.Items.Add("None");
239 
240                 foreach (var name in deviceNames)
241                 {
242                     comboBox.Items.Add(name);
243                 }
244 
245                 if (comboBoxIndex > comboBox.Items.Count)
246                 {
247                     comboBox.SelectedIndex = 0;
248                 }
249                 else
250                 {
251                     comboBox.SelectedIndex = comboBoxIndex;
252                 }
253             }
254         }
255 
256         private void Checkbox_Changed(object sender,
    RoutedEventArgs e)
257         {
258             if (sender is CheckBox checkBox)
259             {
260                 int rowIndex = FindRowIndexByContent(
    TranslationGrid, checkBox.Name[checkBox.Name.Length - 1].
    ToString());
261 
262                 if (checkBox.IsChecked == true)
263                 {
264                     GlobalVariables.MIDIrecordings.Add(rowIndex);
265                 }
266                 else
267                 {
268                     GlobalVariables.MIDIrecordings.Remove(rowIndex)
    ;
269                 }
270             }
271         }
272         private void TextBox_PreviewTextInput(object sender,
    TextCompositionEventArgs e)
273         {
274             if (!char.IsDigit(e.Text, e.Text.Length - 1))
```

```
275             {
276                 e.Handled = true;
277             }
278         }
279 
280         // Type Selector Combobox
281         private void typeSelector_SelectionChanged(object sender,
    EventArgs e)
282         {
283             if (sender is ComboBox comboBox)
284             {
285                 int rowIndex = FindRowIndexByContent(
    TranslationGrid, comboBox.Name[comboBox.Name.Length - 1].
    ToString());
286 
287                 UIElement element = GetChildElementAtPosition(
    TranslationGrid, rowIndex, 6);
288                 if (element != null)
289                 {
290                     TranslationGrid.Children.Remove(element);
291                 }
292 
293                 if (comboBox.SelectedValue.ToString() == "None" ||
    comboBox.SelectedValue == null)
294                 {
295                     return;
296                 }
297 
298                 if (comboBox.SelectedValue.ToString() == "Keyboard
    Stroke")
299                 {
300                     TextBox keystrokeTextbox = new TextBox();
301                     keystrokeTextbox.Name = "KeyboardText" +
    rowIndex;
302                     keystrokeTextbox.TextChanged +=
    setTranslationsTextbox;
303                     keystrokeTextbox.Width = 120;
304                     keystrokeTextbox.Height = 25;
305                     keystrokeTextbox.VerticalAlignment =
    VerticalAlignment.Center;
306                     keystrokeTextbox.HorizontalAlignment =
    HorizontalAlignment.Left;
307                     keystrokeTextbox.VerticalContentAlignment =
    VerticalAlignment.Center;
308                     keystrokeTextbox.HorizontalContentAlignment =
    HorizontalAlignment.Left;
309 
```

```
310                     Grid.SetColumn(keystrokeTextbox, 6);
311                     Grid.SetRow(keystrokeTextbox, rowIndex);
312                     TranslationGrid.Children.Add(keystrokeTextbox);
313                 }
314 
315                 if (comboBox.SelectedValue.ToString() == "Windows
    Parameter")
316                 {
317                     ComboBox windowsCombo = new ComboBox();
318                     windowsCombo.Name = "WindowsCombo" + rowIndex;
319                     windowsCombo.Items.Add("None");
320                     windowsCombo.Items.Add("Volume");
321                     windowsCombo.SelectedIndex = 0;
322                     windowsCombo.Width = 120;
323                     windowsCombo.Height = 25;
324                     windowsCombo.SelectionChanged +=
    setTranslationsCombo;
325                     windowsCombo.VerticalAlignment =
    VerticalAlignment.Center;
326                     windowsCombo.HorizontalAlignment =
    HorizontalAlignment.Left;
327                     windowsCombo.VerticalContentAlignment =
    VerticalAlignment.Center;
328                     windowsCombo.HorizontalContentAlignment =
    HorizontalAlignment.Left;
329 
330                     Grid.SetColumn(windowsCombo, 6);
331                     Grid.SetRow(windowsCombo, rowIndex);
332                     TranslationGrid.Children.Add(windowsCombo);
333                 }
334 
335             }
336         }
337 
338         // Utils
339         private static int FindRowIndexByContent(Grid grid, string
    content)
340         {
341             // Iterate through the children of the grid
342             foreach (UIElement child in grid.Children)
343             {
344                 // Check if the child is a TextBlock and its
    content matches
345                 if (child is TextBlock textBlock && textBlock.Text
    == content)
346                 {
347                     // Get the row index of the child
```

```
348                     int rowIndex = Grid.GetRow(child);
349                     return rowIndex;
350                 }
351             }
352             return -1; // Content not found
353         }
354 
355         private static UIElement GetChildElementAtPosition(Grid
    grid, int row, int column)
356         {
357             foreach (UIElement child in grid.Children)
358             {
359                 int childRow = Grid.GetRow(child);
360                 int childColumn = Grid.GetColumn(child);
361                 if (childRow == row && childColumn == column)
362                 {
363                     return child;
364                 }
365             }
366             return null; // Element not found at the specified
    position
367         }
368 
369         private void setTranslationsTextbox(object sender,
    TextChangedEventArgs e)
370         {
371             setTranslations();
372         }
373 
374         private void setTranslationsCombo(object sender,
    SelectionChangedEventArgs e)
375         {
376             setTranslations();
377         }
378 
379         private void setTranslationsCombo(object sender, EventArgs
    e)
380         {
381             setTranslations();
382         }
383 
384         private void setTranslations()
385         {
386             if (loaded == false)
387             {
388                 return;
389             }
```

```
390
391             Dictionary<int, List<string>> translations = [];
392
393             for (int i = 1; i != GlobalVariables.translationCount +
     1; i++)
394             {
395                 List<string> translationData = [];
396
397                 UIElement element = GetChildElementAtPosition(
    TranslationGrid, i, 1);
398                 if (element is ComboBox comboBox)
399                 {
400                     translationData.Add(comboBox.SelectedValue.
    ToString());
401                 }
402
403                 element = GetChildElementAtPosition(TranslationGrid
    , i, 3);
404                 if (element is TextBox textBox)
405                 {
406                     translationData.Add(textBox.Text);
407                 }
408
409                 element = GetChildElementAtPosition(TranslationGrid
    , i, 4);
410                 if (element is ComboBox comboBox1)
411                 {
412                     translationData.Add(comboBox1.SelectedValue.
    ToString());
413                 }
414
415                 element = GetChildElementAtPosition(TranslationGrid
    , i, 6);
416                 if (element is TextBox textBox1)
417                 {
418                     try
419                     {
420                         translationData.Add(textBox1.Text);
421                     }
422                     catch (Exception) { throw; }
423                 }
424                 if (element is ComboBox comboBox2)
425                 {
426                     if (comboBox2.SelectedValue != null)
427                     {
428                         translationData.Add(comboBox2.SelectedValue
    .ToString());
```

```
429                         }
430                     }
431 
432                     translations.Add(i,translationData);
433                     System.Diagnostics.Debug.WriteLine(string.Join(",
    ", translationData));
434                 }
435 
436 
437                 GlobalVariables.translations = translations;
438             }
439         }
440 }
```