Design Document

MXLify

EECS 2311 - Group 9

Zach Ross
Christopher Moon
Erika Grandy
Yasser Al Zahed
Faruq Afolabi

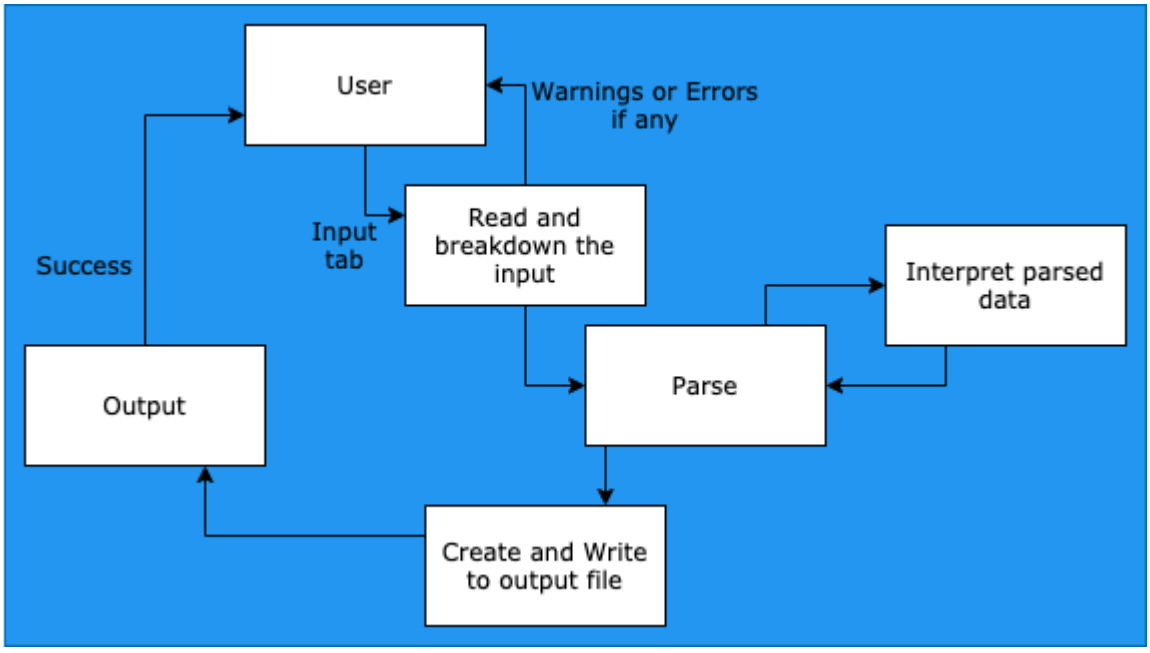
04/13/2021

Table of Contents

1. Introduction

This document gives a high level look into the design of the MXLify software. It gives a glimpse into the different classes that come together in order to convert guitar, drum or bass tablature into a MusicXML file that can then be used by a variety of professional music software.

This is a high level diagram that displays the flow of the systems within the MXLify software.



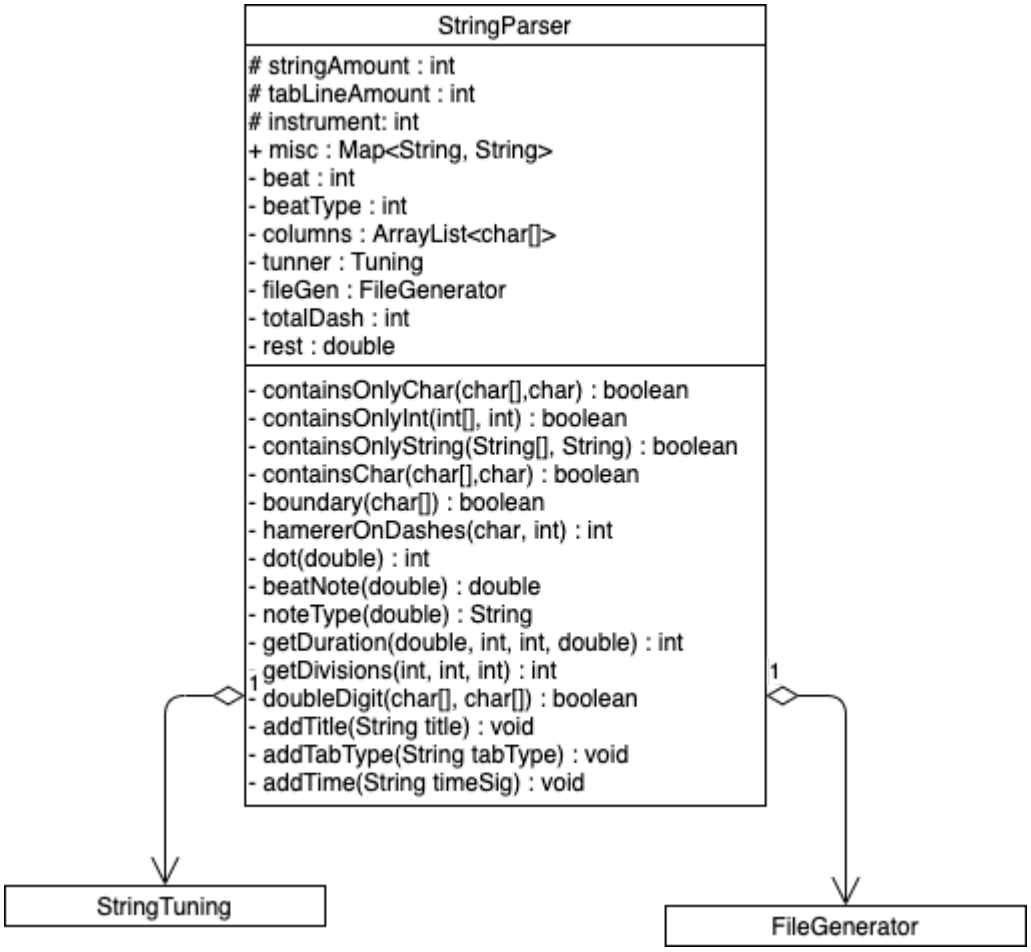
The document also shows how different tasks in the system are carried out using sequence diagrams. These diagrams show how the user input is received and then how the code flows throughout the system to complete the desired result.

The document also gives some maintenance scenarios that give insight on where to look in case there is a need to add more features or fix issues with the current system. If there is a need to do any of this in the future, this document can be used as a point of reference inorder to help execute these changes.

2. Class Diagrams

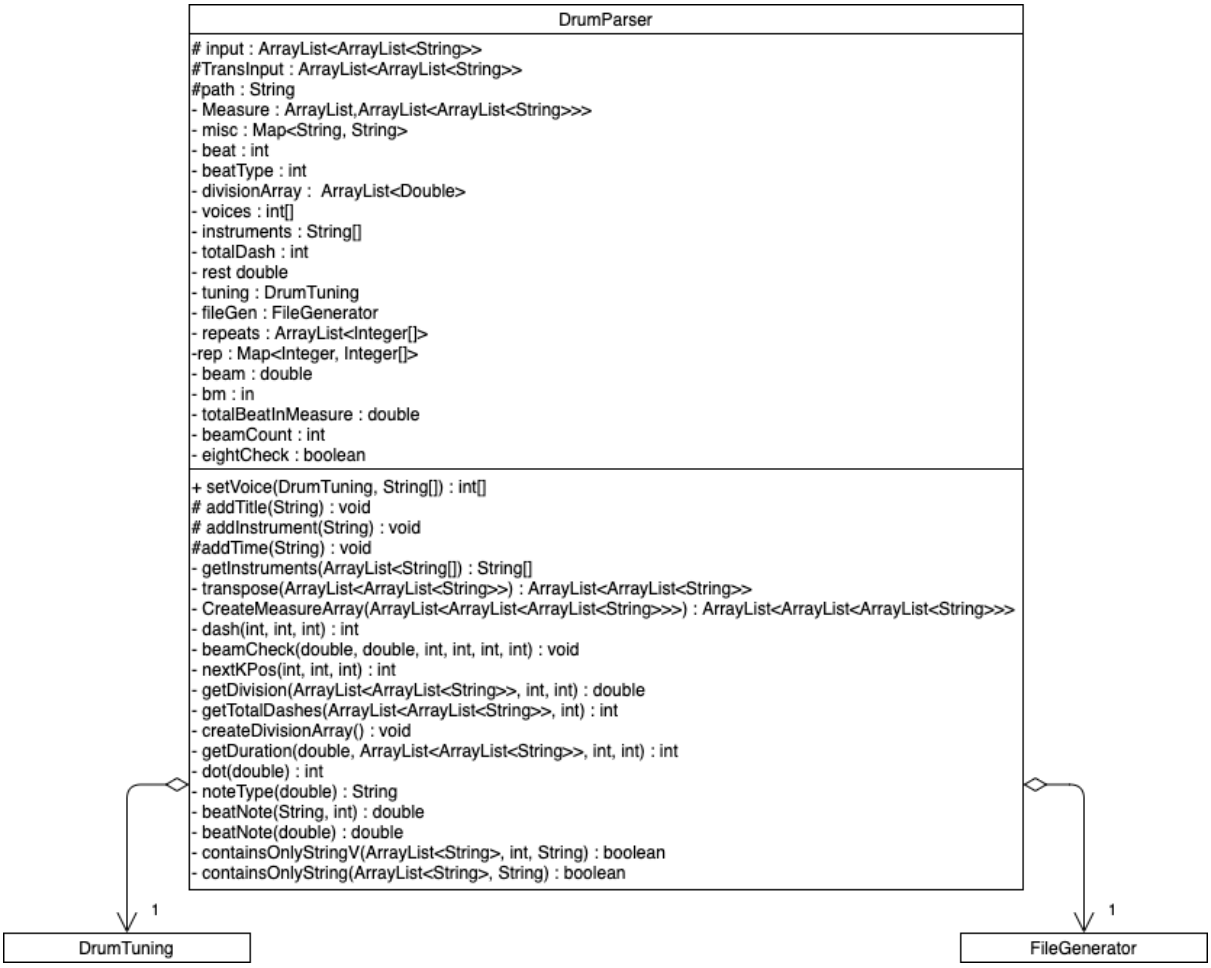
2.1 StringParser:

The StringParser class is used to parse the inputted tablature data. It has a tuning class object that is used to get the string tuning. It has a FileGenerator class object that is used to generate the MusicXML file.



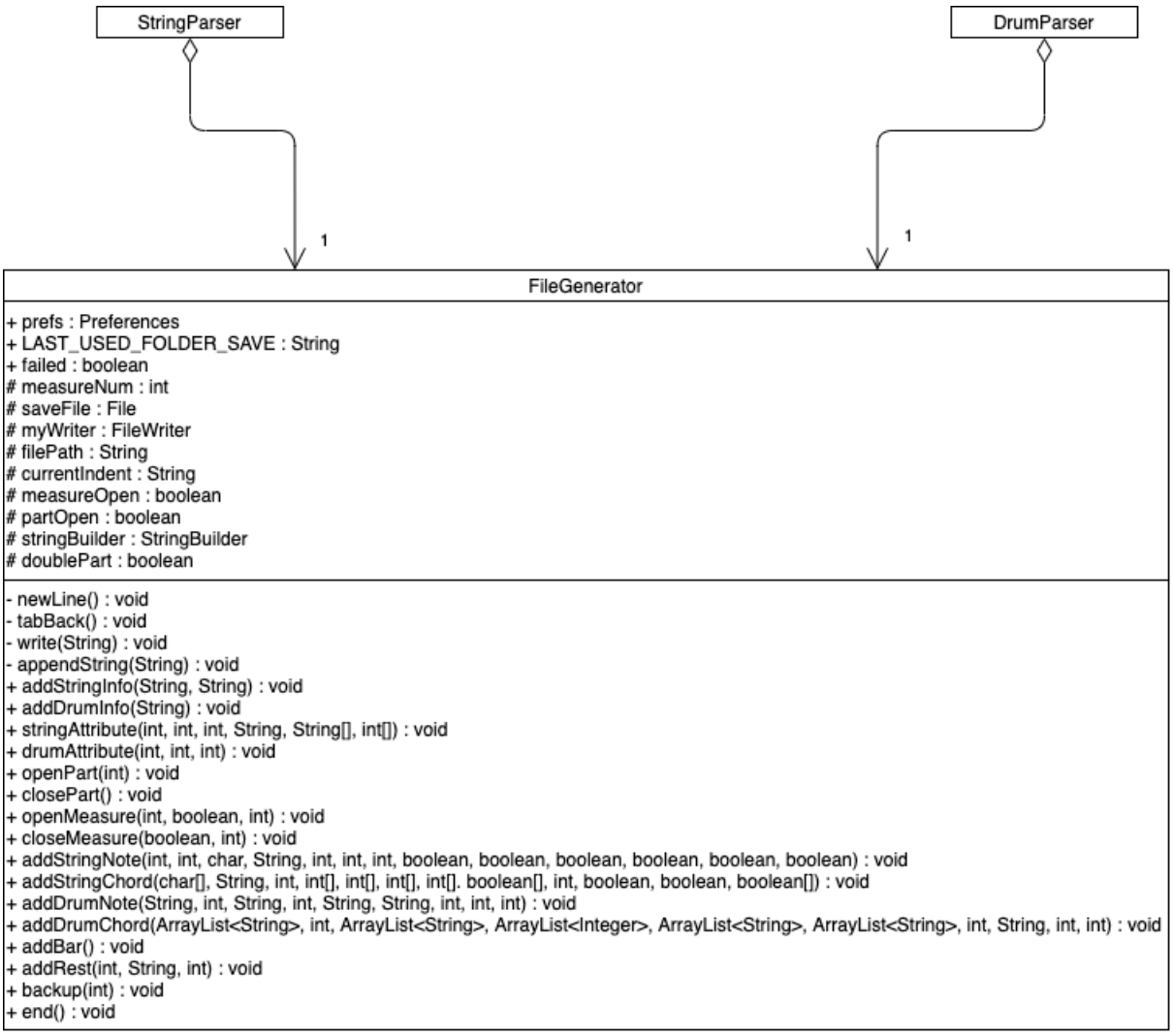
2.2 DrumParser:

The *DrumParser* class is used to parse the inputted tablature data. It has a *tuning* class object that is used to get the drum tuning. It has a *FileGenerator* class object that is used to generate the MusicXML file.



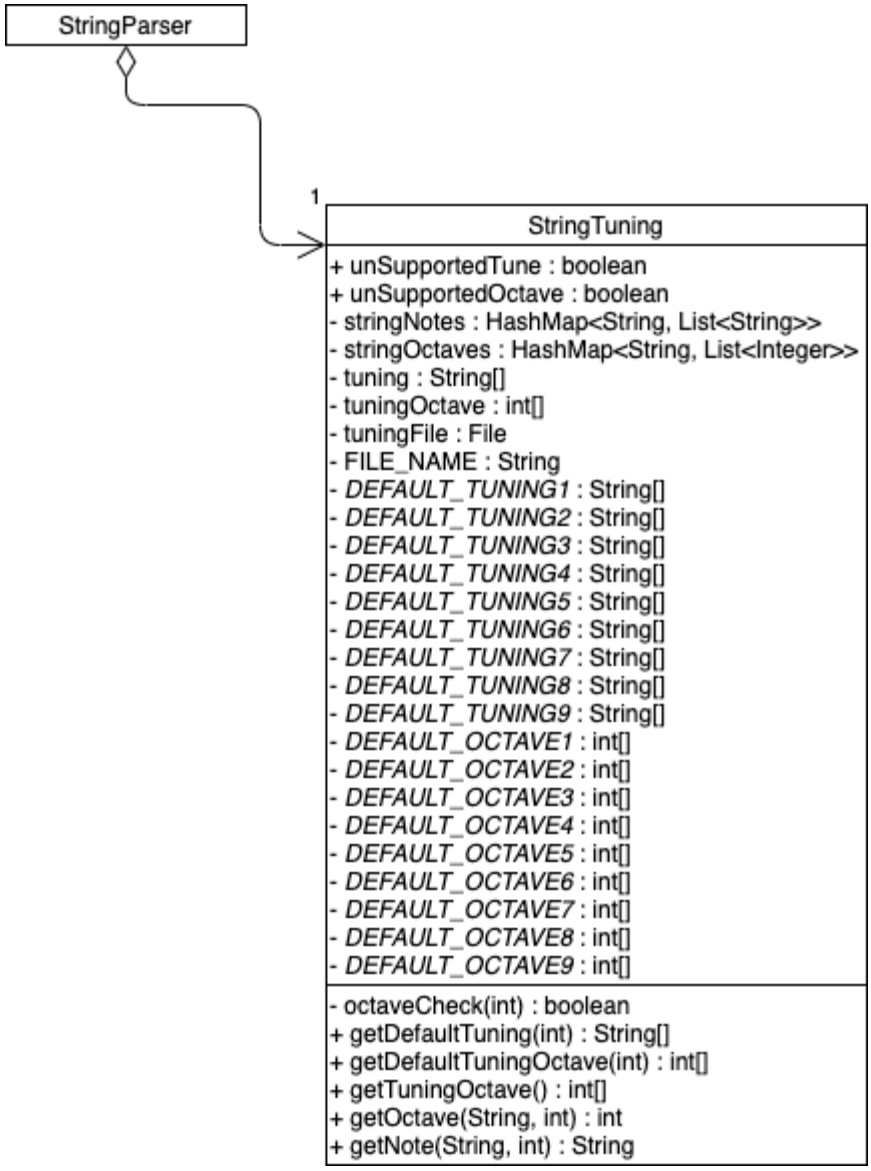
2.3 FileGenerator:

The *FileGenerator* class is used by the *Parser* class to generate the MusicXML file.



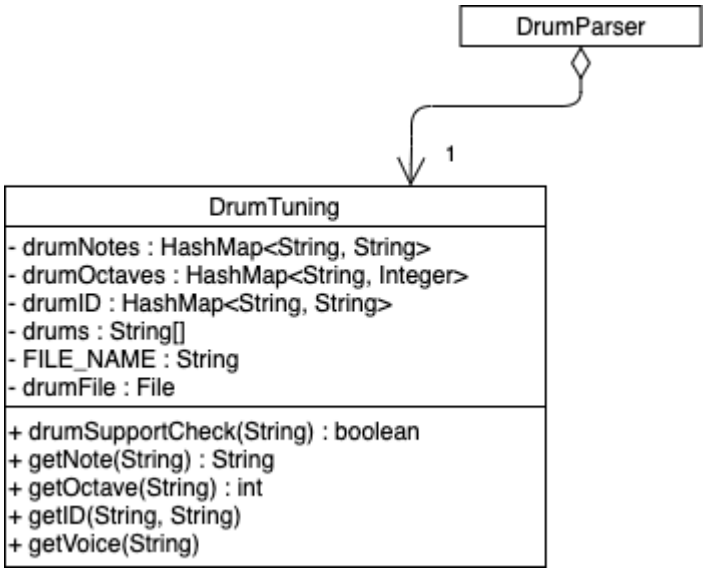
2.4 StringTuning:

The *StringTuning* class is used to get the correct tuning for the inputted guitar or bass tablature.



2.5 DrumTuning:

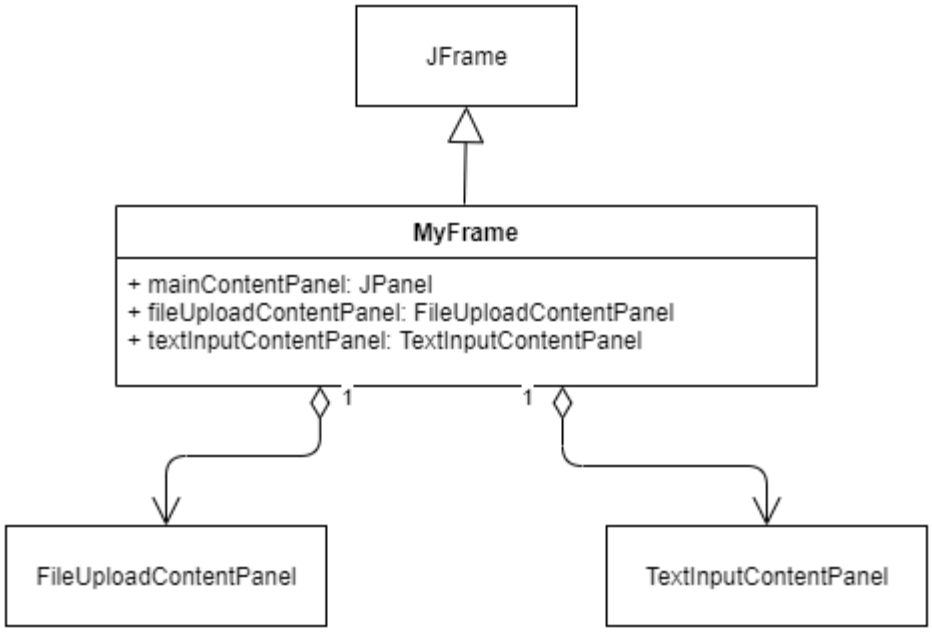
The *DrumTuning* class is used to get the correct tuning for the inputted drum tablature.



2.6 GUI:

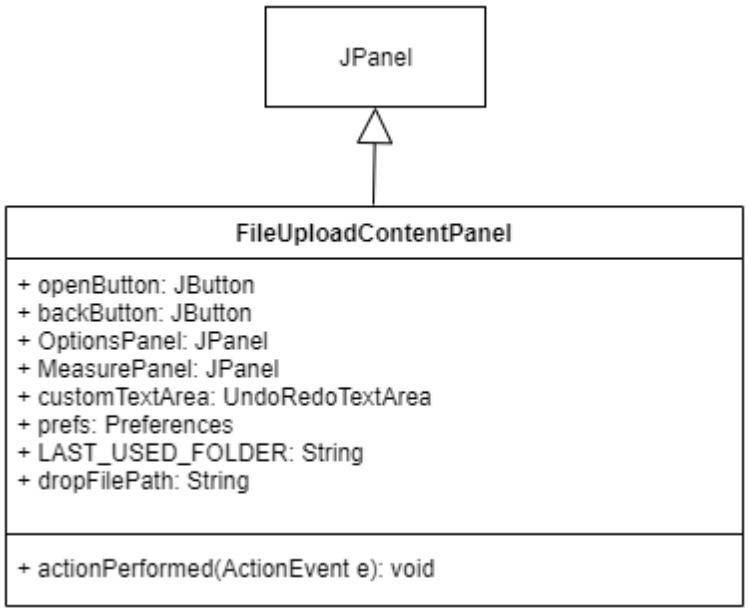
2.6.1 MyFrame:

The *MyFrame* class holds all the GUI elements



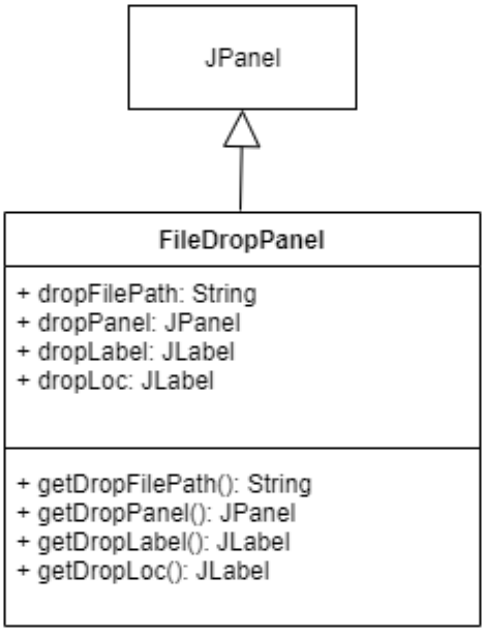
2.6.2 FileUploadContentPanel:

The *FileUploadContentPanel* class holds the file uploading GUI elements



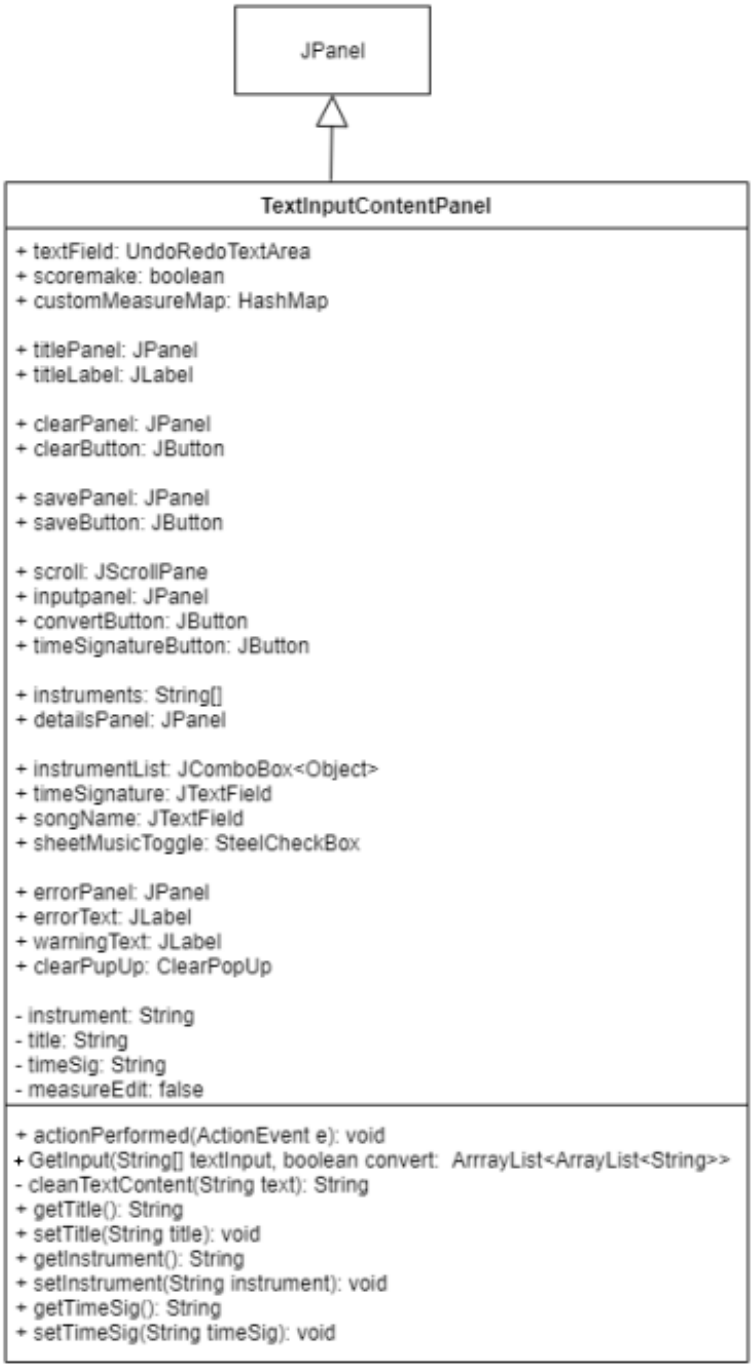
2.6.3 FileDropPanel:

The *FileDropPanel* class holds the text file drag and drop GUI elements



2.6.4 TextInputContentPanel:

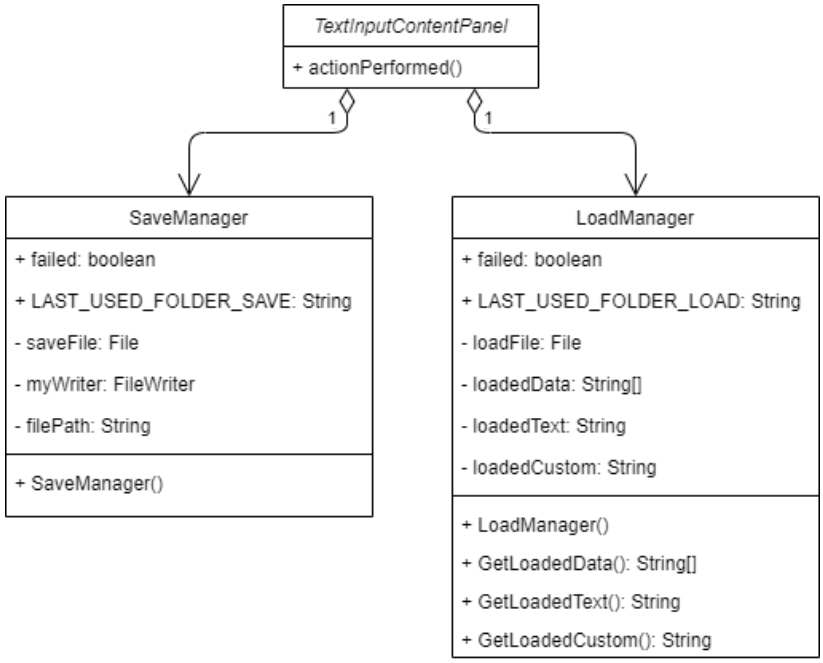
The *TextInputContentPanel* class holds the text input GUI elements



2.7 SaveManager & LoadManager:

The *SaveManager* class is used to save the inputted tablature and metadata to a .mxlify file that can then be loaded.

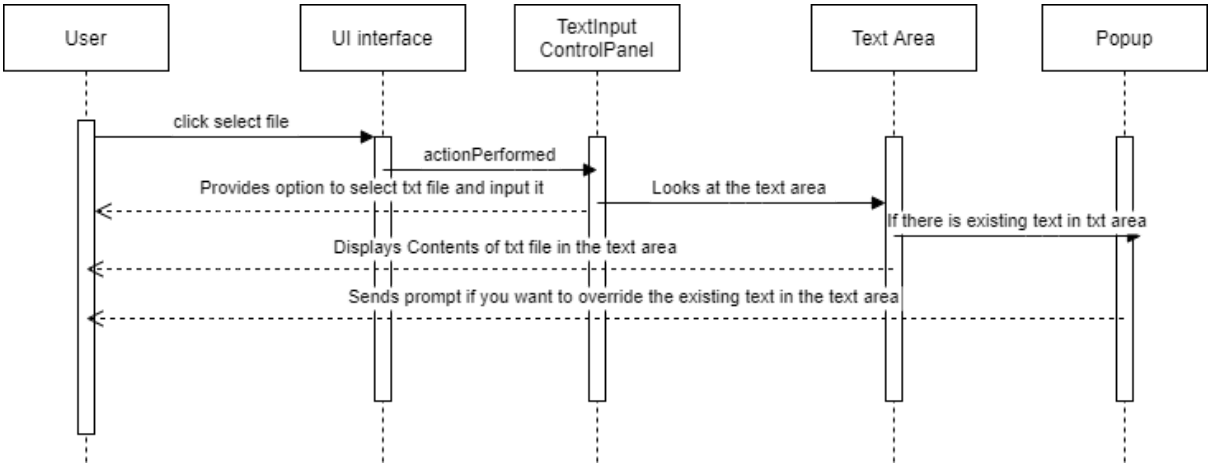
The *LoadManager* class is used to load a .mxlify file that contains saved tablature and metadata.



3. Sequence Diagrams

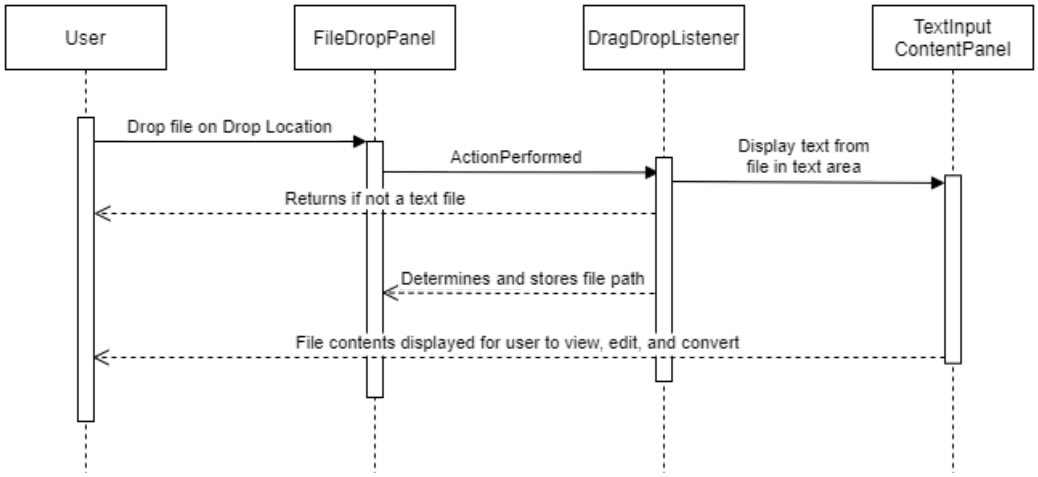
3.1 Select Tablature File:

The user selects a tablature text file to be inputted into the software.



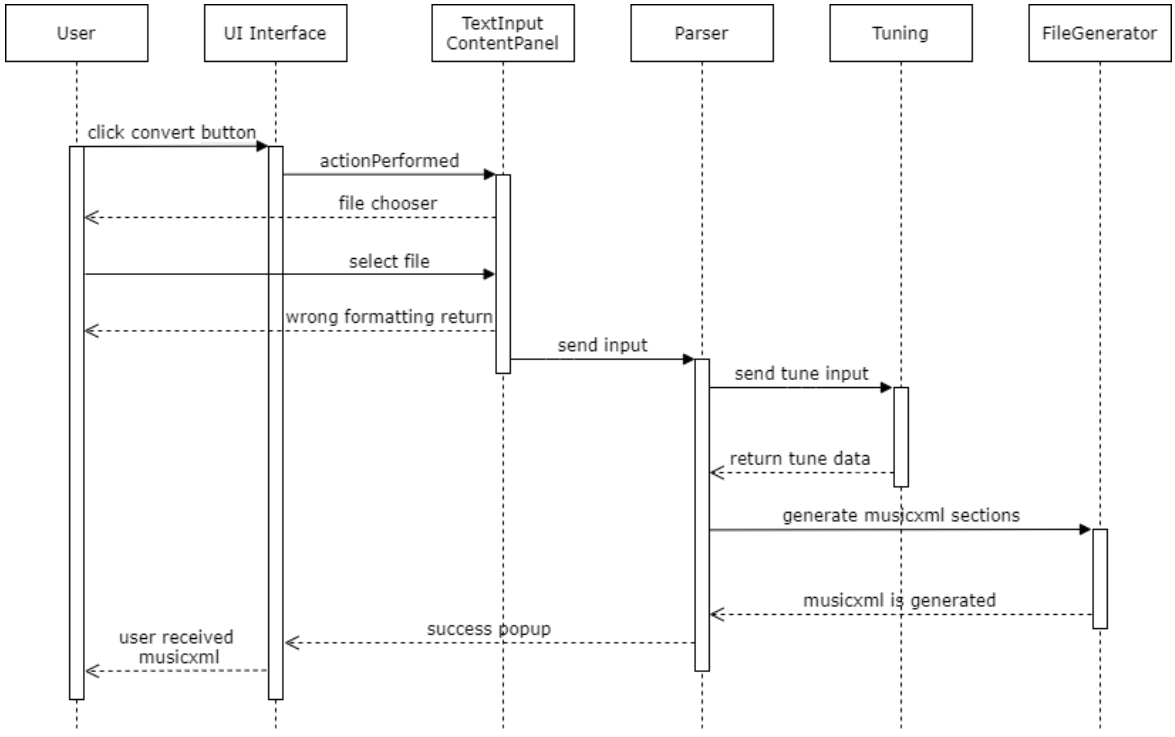
3.2 Drag And Drop Tablature File:

The user drags and drops a tablature text file to be inputted into the software.



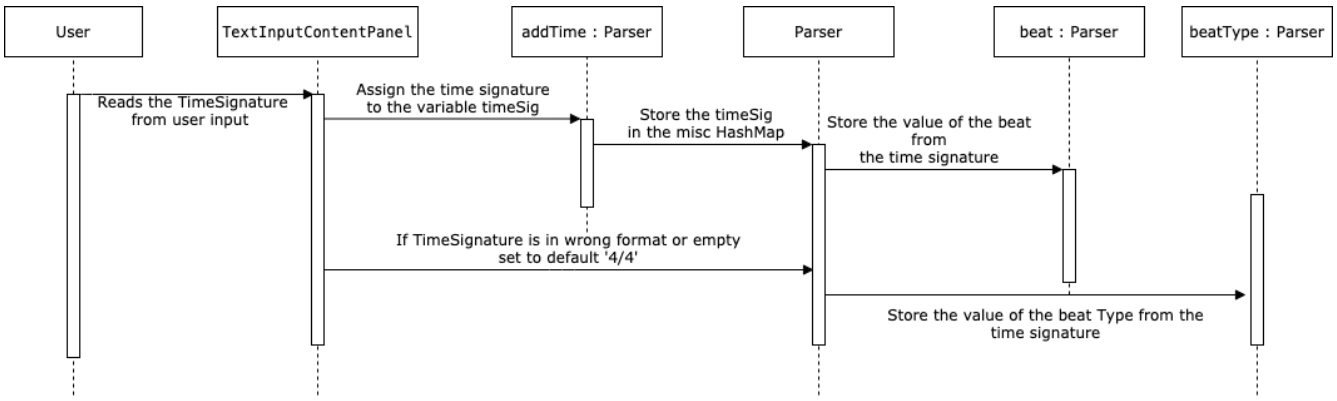
3.3 Convert To MusicXML:

The user clicks the convert button to convert the imputed tablature and output a musicxml file.



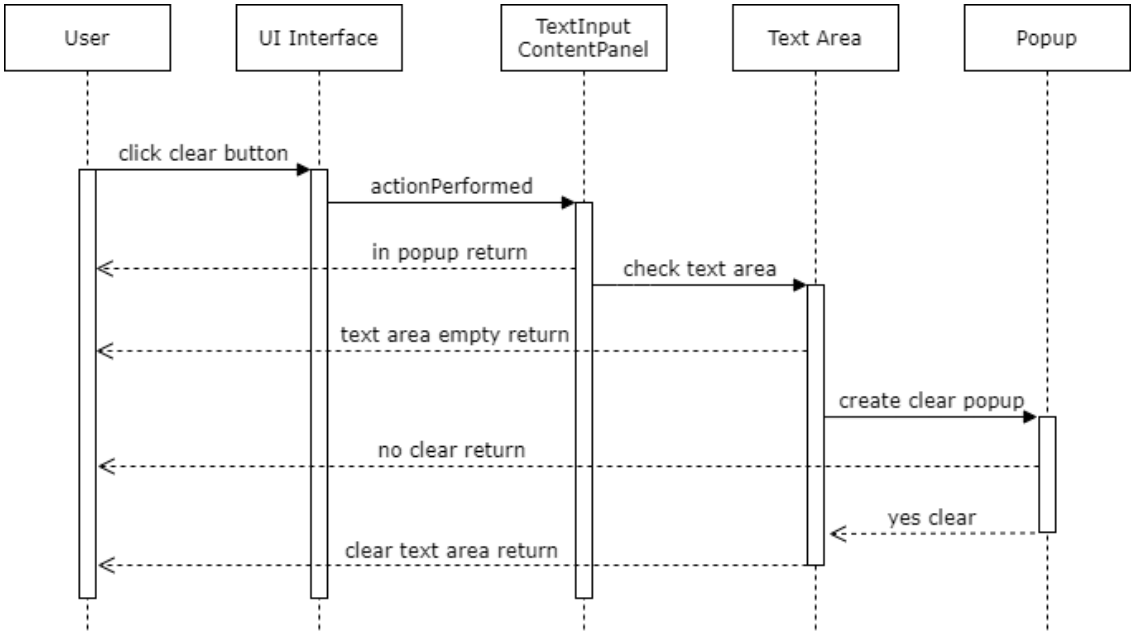
3.4 Set Time Signature:

The user inputs the time signature and the software saves it to be used.



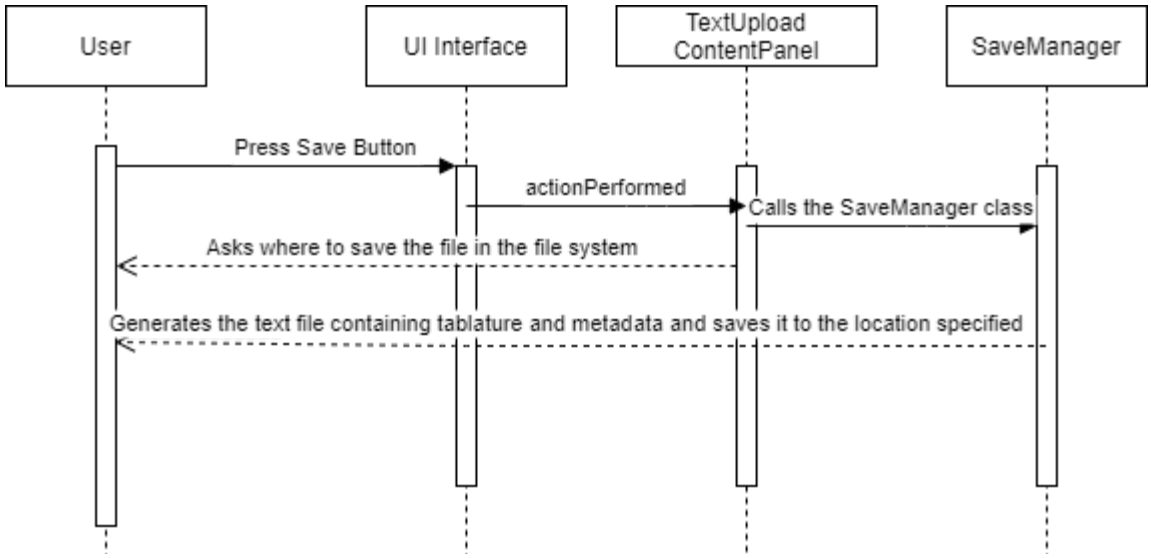
3.4 Clear Text Area:

The user clicks the clear button to clear the text area.



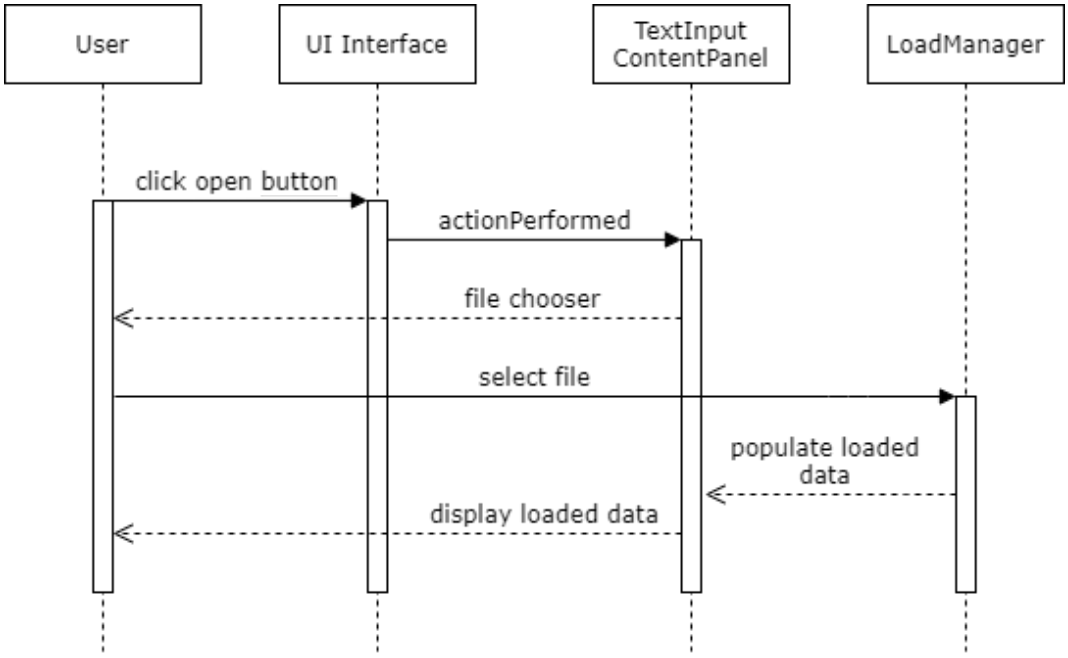
3.5 Save:

The user clicks the save button and chooses where to save the .mxlify file.



3.6 Load:

The user clicks the open button and selects the .mxlify file to be loaded into the software



4. Maintenance Scenarios

4.1 Adding Features

4.1.1 Adding Guitar/Bass Features

To add guitar or bass features changes will need to be made to the *StringParser* class in order to parse the new feature. Currently all tab characters within measures are iterated through, and methods are called in the *FileGenerator* class to generate the MusicXML.

Within the *StringParser* class, a check for any new character type must be added, such as for slide (/) or pull off (p) characters. New methods would need to be created within the *FileGenerator* class, to correctly write these different options to the outputted MusicXML file.

4.1.2 Adding Drum Features

Adding drum features requires additional checks to be added within the *DrumParser* class. Currently all tab characters within measures are iterated through, and a chord or note is added using *addDrumNote()* and *addDrumChord()* from the *FileGenerator* class.

Within the *DrumParser* class, a check for the new character type must be added, such as for flam (f) or drag (d) characters. New methods would need to be created within the *FileGenerator* class, to correctly write these different options to the outputted MusicXML file.

4.1.3 Adding Another Instrument

To add another instrument, you would need to make a new parser class for the instrument you are adding, that would parse the contents of the tablature for that instrument. Within the *InstrumentDetection* class code will need to be added in order to detect the newly added instrument. It returns an integer that is used to run the appropriate parser class (0,1,2 are already used). Within the *Main* class *Convert()* method you would need to add another case to the switch statement that will call the newly added instrument parser. Within the file generator you can create methods that take parameters passed from the newly added parser to generate any new MusicXML that is needed.

4.2 Fixing Issues

4.2.1 Fixing String Tuning Issues

If there is an issue with the tuning that needs to be fixed, debugging needs to be done in the *StringTuning* class in order to find the issue. You may also need to debug the *parser* class for any issues that might be related to tuning.

4.2.2 Fixing String Parsing Issues

If there are any issues with the parsing of string input, debugging needs to be done in the *StringParser* class in order to make sure the input is being parsed properly. You may need to also debug the *TextInputContentPanel* class in order to make sure the input is getting past the error checks that are in place.

4.2.3 Fixing Drum Tuning Issues

If there is an issue with the drum tuning that needs to be fixed, debugging needs to be done in the *DrumTuning* class in order to find the issue. You may also need to debug the *DrumParser* class for any issues that might be related to tuning.

4.2.4 Fixing Drum Parsing Issues

If there are any issues with the parsing of drum input, debugging needs to be done in the *DrumParser* class in order to make sure the input is being parsed properly. You may need to also debug the *TextInputContentPanel* class in order to make sure the input is getting past the error checks that are in place.

4.2.5 Fixing Save/Load Issues

If there is an issue with saving or loading that needs to be fixed, debugging needs to be done in the *SaveManager* class or *LoadManager* class. You may need to also debug the *TextInputContentPanel* class to make sure the data is being passed and received correctly from the *SaveManager* class or *LoadManager* class.