TAB2XML

Testing Document

Final Version

04/11/2021

Written By:	Ziqi Zhou	214726283
	Hargovind Singh	217303025
	Ali Yamany	216513269
	Amer Alshoghri	214992291
	Uwais Kazi	215263940

Table of Contents

List of Figures

1. Introduction

This Testing Document (TD) provides the information necessary for quality assurance to effectively use TAB2XML final release. This TD also details existing unit tests and current testing coverage.

2. Overview

TAB2MXL is a standalone desktop application designed to convert different formats of ASCII music tablatures to a popular music description file format called MusicXML. This software can convert different types of tablature (guitar, drums, and bass) into platform and instrument independent MusicXML.

TAB2XML provides an easy-to-use user interface compatible with most operating systems. Both file processing and direct text input methods are available to users. Translated MusicXML files are accessible through both the interface and the file output.

2.1 Conventions

The term ‘user’ is used throughout this document to refer to a person who requires and/or has acquired access to the TAB2XML.

The term ‘action’ is used throughout this document to refer to a mouse click on a menu or button, and typing in the text area while interacting with the graphic user interface of TAB2XML.

2.2 Cautions & Warnings

The current release of TAB2XML is only intended for users with access granted by the product owner. This application may not be distributed or referenced without the team’s consent.

3. Design Test Cases

In this section, we will discuss the design and implementation of all test cases implemented. For each test class, the important and essential test cases will be discussed in full detail. Other smaller test cases in that class will be included by name for reference.

3.1 Note Class Test Cases

Note class is a basic data structure in TAB2XML that is used to represent the fundamental element Note in a tablature. Some important elements in a Note object include attributes such as octave, step, and methods such as getter and setters.

noteTestVoice

This test case ensures that the Note class constructor is working as intended. Many other test cases in this class perform similar, simple tests. We included these tests to ensure that any changes to the main Note constructor would not introduce issues into its child class or otherwise cause regressions.

noteTestOctave

Test Note constructor and getOctave()

noteTestStep

Test Note constructor and getStep()

noteTestDuration

Test setDuration(int duration)

noteTestVoiceSetter

Test setVoice(int voice)

testSetOctave

Test setOctave(int octave)

testSetColumn()

Test setColumn(int column)

testSetStep()

Test setStep(String step)

testSetString()

Test setString(int string)

testSetFret()

Test setFret(int fret)

testSetType()

Test setType(String type)

testAlter()

Test setAlter(int alter)

testConstructorAlter()

Test constructor Note(String step, int alter)

testConstructorAlterOctave()

Test constructor Note(String step, int alter, int octave)

3.2 DrumNote Class Test Cases

The DrumNote class is a basic data structure in TAB2XML that is used to represent the fundamental element Note in a drum tablature. DrumNote extends Note class. Some important elements in a DrumNote object include attributes such as notehead, instrumentId, and instrumentName to distinguish it from the parent class.

testSetInstrument()

This and similar test cases are focused on testing the validity of the DrumNote constructor. this constructor needed to be tested as the class is further specialized from its parent class, and we wanted to ensure that DrumNote objects did not undergo any regressions should changes be made to the constructor.

testSetNotehead()

Test setNoteHead(String notehead)

testSetInstrumentName()

Test setInstrumentName(String instrumentName)

noteTestVoice()

Test getVoice() working as expected after extending Note class

noteTestOctave()

Test getOctave() working as expected after extending Note class

noteTestStep()

Test getStep() working as expected after extending Note class

noteTestDuration()

Test setDuration(int duration) and getDuration() working as expected after extending Note class

noteTestVoiceSetter()

Test setVoice(int voice) and getVoice() working as expected after extending Note class

testSetOctave()

Test setOctave(int octave) working as expected after extending Note class

testSetColumn()

Test setColumn(int column) and getColumn() working as expected after extending Note class

testSetStep()

Test setStep(String step) working as expected after extending Note class

testSetString()

Test setString(int string) working as expected after extending Note class

testSetFret()

Test setFret(int fret) working as expected after extending Note class

testSetType()

Test setType(String type) working as expected after extending Note class

testSecondConstructor()

Test DrumNote(String step, int octave, String instrumentId, String instrumentName)

3.3 Measure Class Test Cases

Measure class is a basic data structure in TAB2XML that is used to represent the fundamental element Measure in a tablature. Measure class contains Note objects. Some important elements in a Measure object include attributes such as measureList, division, and methods such as getter and setters.

testGetMeasureNumber()

This test case tests the functionality of getMeasureNumber(), gettimeBeatType(), and getTimeBeats(). It's very important to ensure that the measure's indexing value is preserved throughout the different operations it undergoes as we parse different tablatures and populate the Measure objects. It's critical that the ordering remains intact as we use the measure index for several other important functions like beam generation and repeat detection.

testNoteList()

This test case tests the basic functionality of the note list attribute in each measure. Note lists must retain a specific order for the XML generation to function correctly. This ensures that any changes made to the note list do not affect the ordering of notes, which would cause issues in the XML generation.

testGetClefLine()

Test getClefLine()

testGetDivisions()

Test constructor Measure(int division) and getDivision()

3.4 NoteUtility Test Cases

NoteUtility class is a utility class in TAB2XML that is used to assist with DrumNote and Note parsing. NoteUtility class contains important attributes such as guitarNote and drumNote, and it contains methods such as makeArray and initialise.

testDrumInitialize()

The initializeDrum() method tests the setup of the DrumNote HashMap. This is the data structure that stores all the drum notes and their respective octaves, and the “tuning” fields required for XML generation. This initialise function is responsible for initially setting the drum note values.

testGuitarInitialize()

The testGuitarInitialize() method tests the setup of the guitar note array. This is the data structure that stores all the guitar notes and all relevant information for each note. The tuning of the guitar is also stored here. The initialise function that this tests is responsible for both initially setting the note values, as well as updating our tuning should the user choose to specify their own.

testNoteType()

Test if getNoteType(float type) method properly retrieve the noteType based on a float input

testCounterGetterSetter()

Test if counter and octave getters and setters work properly

3.5 BassNoteUtility Test Cases

BassNoteUtility class is a utility class in TAB2XML that is used to assist with Bass tablature parsing. NoteUtility class contains important attributes such as bassNote, and it contains methods such as makeArray and initialise.

testInitialize()

This test case tests if initialise() method properly sets up the bassNote 2D array based on either default or user tuning. Since the initialise() method dynamically sets the 2D array, we want to make sure that it is set up properly. We perform some simple tuning before initialising the BassNoteUtility class, then we compare the generated 2D array to our expected 2D array.

testCounterGetterSetter()

Test if counter and octave getters and setters work properly

3.6 StringParserUtility Test Cases

StringParserUtility is a utility class in TAB2XML that is used to parse input guitar tablature. StringParserUtility includes important static attributes such as stringParse, and it contains methods such as generate, getDivision, and getNote.

testSimple()

This test case tests if the StringParserUtility class properly parses a simple guitar tablature. After confirming the correctness of the XmlGenerator class, by passing a simple guitar tablature to the StringParserUtility and observing its output will ensure the correctness of basic guitar translation capability. We generated the expected output by carefully examining our XmlGenerator class and utility class. We ensure the expected output is correct, then we compare it to the output generated by our application when the same guitar tablature is passed.

resetOctaveTest()

This test case tests if the StringParserUtility class properly applies user tuning. After confirming that a simple translation works with our application, we hope that our users can properly tune their guitar tablature if they wish. We use the same guitar tablature as testSimple() since the general structures remain the same. We then apply tuning. We ensure our expected result is correct, then we compare our expected result to the actual result to ensure our StringParserUtility class is applying the tuning properly.

testComplex()

This test case tests if the StringParserUtility class properly translates grace notes. We use a simple guitar tablature containing grace notes as input. We carefully examine our classes and generate an expected result. We make sure the expected result is correct. We then compare the expected result to the actual result to ensure our StringParserUtility class is identifying and translating grace notes correctly.

testRepeat()

Test if the StringParserUtility class properly translates tablatures with repeated measures.

3.7 StringParserUtilityDrum Test Cases

StringParserUtilityDrum is a utility class in TAB2XML that is used to parse input drum tablature. StringParserUtilityDrum extends StringParserUtility. It includes important static attributes such as measureList, and it contains methods such as stringParse, getDivision, and getNote.

testDrumSimple()

After confirming the correctness of the XmlGenerator class, by passing a simple drum tablature to the StringParserUtilityDrum and observing its output will ensure the correctness of basic drum translation capability. We generated the expected output by carefully examining our XmlGenerator class and utility class. We ensure the expected output is correct, then we compare it to the output generated by our application when the same drum tablature is passed.

3.8 StringParserUtilityBass Test Cases

StringParserUtilityBass is a utility class in TAB2XML that is used to parse input drum tablature. StringParserUtilityBass extends StringParserUtility. It includes important static attributes such as measureList, and it contains methods such as stringParse, getDivision, and getNote.

testBass()

After confirming the correctness of the XmlGenerator class, by passing a simple bass tablature to the StringParserUtilityBass and observing its output will ensure the correctness of basic bass translation capability. We generated the expected output by carefully examining our XmlGenerator class and utility class. We ensure the expected output is correct, then we compare it to the output generated by our application when the same bass tablature is passed.

3.9 XmlGenerator Test Cases

XmlGenerator is a utility class in TAB2XML that is used to generate MusicXML output based on a list of measures. It includes important static methods such as generate.

testSimpleTranslation()

This test class tests if XmlGenerator class properly translates a simple measure list for guitar. The XmlGenerator class takes in one ArrayList of measures and returns MusicXML content. By calling the generate(ArrayList<Measure> measureList> method, most important private methods in XmlGenerator will be tested through testing the output. We generate the correct output by translating the input Measure list properly. We then compare the actual output to the expected output.

testException()

Test if an exception is thrown when the input measure list is empty

testSimpleDrum()

Test if XmlGenerator class properly translate a simple measure list for drum

testSimpleBass()

Test if XmlGenerator class properly translate a simple measure list for bass

testComposerTitle()

Test if XmlGenerator class properly includes user-defined title and composer

4. User Interface Test Cases

4.1 Application Test Cases

ApplicationTest class uses testFX to test the behaviours of the Primary Stage.

bothButtonEnabled(FxRobot robot)

This test case checks if both translate and save buttons are enabled when the input text area is not empty. The FxRobot clicks on the text area to make sure the text area is in focus. Then the FxRobot writes “hi” in the text area. The FxRobot then observes that both the translate and save buttons are enabled.

should_contain_translate_button(FxRobot robot)

Check if translate button is on the screen

should_contain_save_button(FxRobot robot)

Check if the save button is on the screen

bothButtonDisabled(FxRobot robot)

Check if both translate and save buttons are disabled when the input text area is empty

restartButtonDisabled(FxRobot robot)

Check if the restart button is disabled upon launching the application

fileButtonEnabled(FxRobot robot)

Check if the file button is enabled upon launching the application

4.2 Controller Test Cases

ControllerTest class tests static methods of the controller.

testCleanUp()

The cleanup method in Controller deletes lines that do not contain “|” and “-”. We use a simple tablature containing invalid lines and compare our expected result to the actual output. This test case checks if the cleanup method ignores invalid lines of the tablature.

testRepeatCleanup()

Check if the repeatCleanUp method ignores REPEAT in a tablature

4.3 Translation Options Test Cases

TranslationOptionsTest class uses testFX to test the behaviours of the Translation Options window.

should_contain_confirm(FxRobot robot)

Check if the translation options screen has the confirm button

should_contain_confirm(FxRobot robot)

Check if the translation options screen has the cancel button

4.4 Tuning Options Test Cases

TuningOptionTest class uses testFX to test the behaviours of the Tuning Options window.

has_confirm_button(FxRobot robot)

Check if the tuning options screen has the confirm button

has_cancel_button(FxRobot robot)

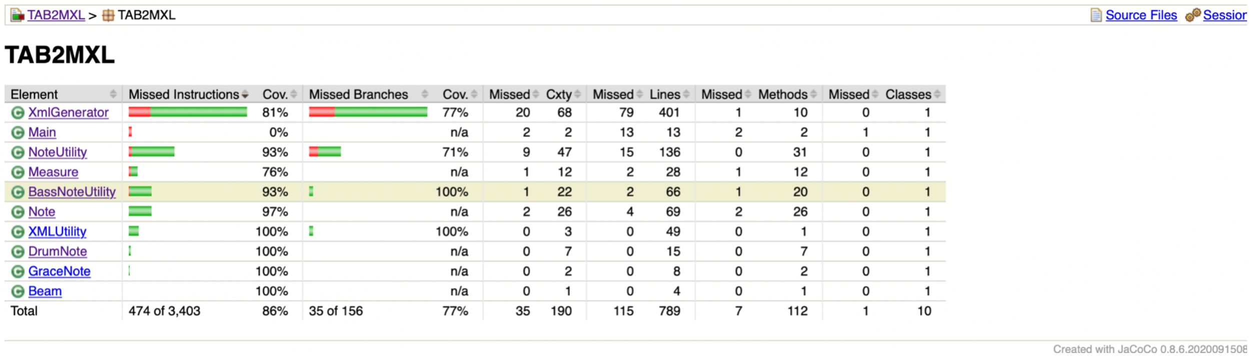
Check if the tuning options screen has the cancel button

5. Testing Coverage

5.1 Coverage

Coverage for Design

Figure 1 - coverage of TAB2XML design



TAB2MXL > TAB2MXL Source Files Sessior

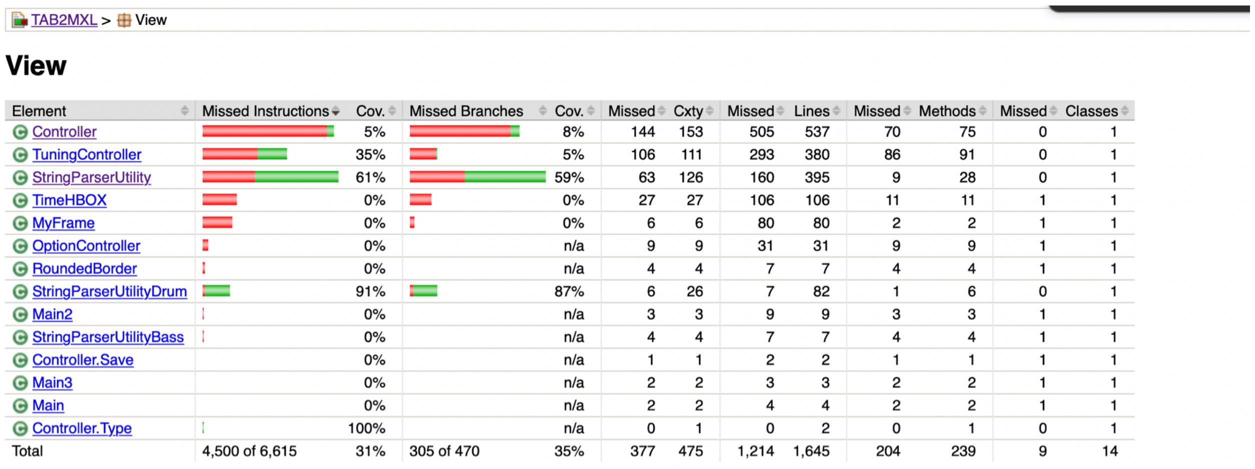
TAB2MXL

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
XmlGenerator		81%		77%	20	68	79	401	1	10	0	1
Main		0%		n/a	2	2	13	13	2	2	1	1
NoteUtility		93%		71%	9	47	15	136	0	31	0	1
Measure		76%		n/a	1	12	2	28	1	12	0	1
BassNoteUtility		93%		100%	1	22	2	66	1	20	0	1
Note		97%		n/a	2	26	4	69	2	26	0	1
XMLUtility		100%		100%	0	3	0	49	0	1	0	1
DrumNote		100%		n/a	0	7	0	15	0	7	0	1
GraceNote		100%		n/a	0	2	0	8	0	2	0	1
Beam		100%		n/a	0	1	0	4	0	1	0	1
Total	474 of 3,403	86%	35 of 156	77%	35	190	115	789	7	112	1	10

Created with JaCoCo 0.8.6.2020091508

Coverage for User Interface

Figure 2 - coverage of TAB2XML interface



TAB2MXL > View

View

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Controller		5%		8%	144	153	505	537	70	75	0	1
TuningController		35%		5%	106	111	293	380	86	91	0	1
StringParserUtility		61%		59%	63	126	160	395	9	28	0	1
TimeHBOX		0%		0%	27	27	106	106	11	11	1	1
MyFrame		0%		0%	6	6	80	80	2	2	1	1
OptionController		0%		n/a	9	9	31	31	9	9	1	1
RoundedBorder		0%		n/a	4	4	7	7	4	4	1	1
StringParserUtilityDrum		91%		87%	6	26	7	82	1	6	0	1
Main2		0%		n/a	3	3	9	9	3	3	1	1
StringParserUtilityBass		0%		n/a	4	4	7	7	4	4	1	1
Controller.Save		0%		n/a	1	1	2	2	1	1	1	1
Main3		0%		n/a	2	2	3	3	2	2	1	1
Main		0%		n/a	2	2	4	4	2	2	1	1
Controller.Type		100%		n/a	0	1	0	2	0	1	0	1
Total	4,500 of 6,615	31%	305 of 470	35%	377	475	1,214	1,645	204	239	9	14

5.2 Future Improvement

As shown in Figure 1, the statement coverage for design has achieved more than 80 percent. The lack of coverage is mainly due to unused classes and methods. Statement coverage for the user interface is only 31%. The team did not receive clear direction for unit testing using TestFX and many functionalities failed to yield expected results. Many classes such as MyFrame and RoundedBorder are also legacy code for our previous JavaSwing GUI and not covered by our test cases. For future improvement, we hope to improve code coverage for the interface by familiarizing the QA team with TestFX.