MIDIPLAYER

COMP4905 – Honours Project

Author: Yulun Wu

Supervisor: Louis Nel

Organization: Carleton University

Date: December 11, 2017
Abstract

MIDIPlayer is an application I developed on IOS platform for my honours project. It was designed for people who want to play MIDI files on their iPhone or for people who want to create or modify MIDI files. This application can be used to import, play, edit or create MIDI files.
Acknowledgment

I would like to thank my supervisor Professor Louis Nel for the support of my honour project and for his patience and knowledge.
# Table of Contents

Introduction .............................................................................................................................................. 6
Preparation .................................................................................................................................................. 7
Installation and Compilation .................................................................................................................... 8
GUI (Graphic User Interface) .................................................................................................................. 12
User Manual .............................................................................................................................................. 14
  Play a Sample MIDI File ....................................................................................................................... 14
  Creating a New MIDI File and Composing a song ............................................................................... 17
  Rename a saved file .............................................................................................................................. 21
  Delete a saved file ............................................................................................................................... 22
  Import MIDI Files ............................................................................................................................... 23
Methodology ............................................................................................................................................ 24
  Create a New Project ......................................................................................................................... 24
  Table View Controller Scene ............................................................................................................. 24
  Changing Scenes ............................................................................................................................... 29
Player Controller View Scene ................................................................................................................ 31
  Play and Reset .................................................................................................................................. 32
  Show the List of Notes in a Text View ............................................................................................... 37
  Change a track’s instrument ............................................................................................................... 40
  Save the MIDI File ............................................................................................................................ 42
  Delete a Track .................................................................................................................................. 43
  Delete a Note .................................................................................................................................... 43
  Import MIDI Files ............................................................................................................................... 44
Results ...................................................................................................................................................... 46
Improvement in the Future ...................................................................................................................... 46
Bibliography ............................................................................................................................................ 47
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>open the project</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>create a provisioning profile</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>build your app</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>main menu of MIDIPlayer</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>storyboard</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>UI controller class files</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>sample MIDI files list</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>Player Controller Scene</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>Playing MIDI file in Player Controller Scene</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>Creating Page View Controller Scene</td>
<td>17</td>
</tr>
<tr>
<td>11</td>
<td>changing track in Player Controller Scene</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>selecting instrument in Player Controller Scene</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>saved MIDI files</td>
<td>20</td>
</tr>
<tr>
<td>14</td>
<td>Rename Page Controller Scene</td>
<td>21</td>
</tr>
<tr>
<td>15</td>
<td>deleting a MIDI file</td>
<td>22</td>
</tr>
<tr>
<td>16</td>
<td>import MIDI file from other application</td>
<td>23</td>
</tr>
<tr>
<td>17</td>
<td>creating Table View Controller Scene</td>
<td>25</td>
</tr>
<tr>
<td>18</td>
<td>creating a new swift file</td>
<td>25</td>
</tr>
<tr>
<td>19</td>
<td>changing the contoller of Table View Controller to Table View Controller</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>adding a segue</td>
<td>29</td>
</tr>
<tr>
<td>21</td>
<td>segue</td>
<td>30</td>
</tr>
<tr>
<td>22</td>
<td>MIDI Player Scene</td>
<td>31</td>
</tr>
<tr>
<td>23</td>
<td>create an action function for a button</td>
<td>32</td>
</tr>
<tr>
<td>24</td>
<td>showing list of note in the text view</td>
<td>37</td>
</tr>
<tr>
<td>25</td>
<td>register a Document Type and an Imported UTIs in the Info.plist file</td>
<td>44</td>
</tr>
</tbody>
</table>
Introduction

This report introduces the application I made in my honours project: MIDIPlayer. This report will present how to use MIDIPlayer, what this application can be used for, how to develop this application and how to improve this application in the future.

MIDIPlayer is an IOS application which can play, edit, create or import MIDI file. MIDIPlayer was designed for people who want to use their iPhone to play MIDI files and want to do some small modifications on the MIDI files.

A computer with macOS High Sierra and an iPhone with IOS 11 was used to develop this application. Xcode was used in this project as the IDE (integrated development environment). Swift is the programming language to use to code in this project. The IOS app frameworks were used in this project are: UIKit, AudioToolBox, AVFoundation and Foundation.

MIDI file is a file that contains multiple MIDI streams with time information for each event. It supports song, sequence, and track structures, tempo, and time signature. (Back, 2003) The advantages of MIDI file include small file size, easy to modify or manipulate notes and being able to change instruments. (amandaghassaei, 2012)

The reason for me to develop this application because an application that can play or edit MIDI file on iPhone is a very useful tool for people who want to edit or play their MIDI file but happen to not have their desktop or laptop computer with them.
Preparation

In this section, I will talk about the requirements of devices, system and software to develop and build this application.

You will need a computer with macOS High Sierra, an iPhone with IOS 11 and a lightning to USB cable.

You will need an apple id account. You can register an apple id account from here:

https://appleid.apple.com/account

Xcode will need to be installed on your computer. It is the IDE (integrated development environment) that will be used to develop and build the application. It can be download from the appstore or from Apple Developer official website. Here is the link for downloading Xcode:

https://developer.apple.com/xcode/

You can download the whole project of MIDIPlayer from this GitHub link:

https://github.com/deadlyrice/MIDIPlayer
Installation and Compilation

After downloading and unzipping the project of MIDIPlayer from GitHub, launch Xcode. Select “File -> Open...” from the menu bar. Navigate to the directory of the unzipped project of MIDIPlayer. Select “MIDIPlayer.xcodeproj” and then click the “Open” button. The screen should look like this:

![Open the project](image)

*Figure 1: open the project*

Before compiling the code, you will need to create a provisioning profile. A provisioning profile is a bundle of configurations, digital signatures or other digital entities. It is downloaded from your apple developer account. You will need to install the provisioning profile on your device before launching your application. Your application will not launch, if the information in the provisioning profile doesn’t match certain criteria. (Apple Inc., 2016b)
To create a provision file, select “MIDIPlayer” under the project navigator on the left side of the Xcode. Select your team account under the Signing section. If there is no any account, select “Add an Account...” to add an account. Change the Bundle Identifier under the Identity section to other string. Please keep the format of the string of Bundle Identifier to “com.MyCompany.MIDIPlayer”, because the bundle ID string must be a in reverse-DNS format (Apple Inc., 2016a) After selecting your account and changing the Bundle ID, wait for few seconds to let Xcode create a provisioning profile on your account. When the process is completed your screen should look like this:

![Figure 2: create a provisioning profile](image)
To compile the code and build the application on your iPhone, first you need to connect your iPhone to your computer with your lightning to USB cable. Unlock your iPhone. Select your iPhone device as the building target on the top left of Xcode. Click and wait for few seconds to let Xcode build the application.

If the application did not launch on your iPhone, tap “Setting -> General -> Profiles or Profiles & Device Management” on your iPhone. Tap your account and tap “Trust ‘your account’”. Then click to build the app again. Your iPhone screen should look like this if you successfully build and launch the app:
Samples

Saved MIDI Files

Create a new MIDI file

rename a MIDI file

delete a MIDI file

Figure 4: main menu of MIDIPlayer
GUI (Graphic User Interface)

In this section, I will introduce the GUI or Graphic User Interface of MIDIPlayer.

The GUI of MIDIPlayer consists of four scenes. They are Table View Controller Scene, Rename Page Controller Scene, Creating Page View Controller Scene and Player Controller Scene. All scenes were created in the storyboard. The storyboard visually represents the user interface of an IOS application. The scenes of contents and the connections between scenes are shown by the storyboard. (Apple Inc., 2013) To view the storyboard of MIDIPlayer, select “Main.storyboard” under the project navigator. Your screen should look like this:

Figure 5: storyboard
Each scene is controlled by a controller. Table View Controller Scene is controlled by Table View Controller. Rename Page Controller Scene is controlled by Rename Page Controller. Creating Page View Controller Scene is controlled by Creating Page View Controller. Player Controller Scene is controlled by MIDIPlayer Controller.

For each controller, I created a corresponding class and a swift class file. You can view the controller class file from the UIController folder under the project navigator.

Figure 6: UI controller class files
User Manual

In this section, I will talk about how to use MIDIPlayer. I will describe the uses and the functions of MIDIPlayer by going through different scenarios.

Play a Sample MIDI File

Suppose a user want to play a sample MIDI file on MIDIPlayer for testing this application. The user taps on the “sample” option of the main menu. The table shows a list of some sample MIDI files and a back option. The texts of the sample MIDI files are coloured in red. The back option is coloured in black.

<table>
<thead>
<tr>
<th>back</th>
<th>12:47 PM</th>
<th>93%</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDI_sample.mid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>meanwoman.mid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>take5.mid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>teddybear.mid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 7: sample MIDI files list*
The user selects a sample MIDI file he wants to play by tapping on the name of that MIDI file. Suppose the user taps on “MIDI_sample.mid”. The app will change the scene to the Player Controller Scene. The text field with “Track #” represents the track that is currently selected. In this case, track 0 is selected. The text view on the screen will show the list of notes in the selected track.

![Player Controller Scene](image)

*Figure 8: Player Controller Scene*
The user taps on the “Play” button. The application starts to play the song of the MIDI file. The “Play” button will become to “Stop” button. Tap on the “Stop” button will stop the app playing the song. On the top right corner of the screen, there are two numbers which show the current position and the length of the song in second. Tap on “Reset” Button will reset the current position to 0 which means the song will be played from the beginning. The text view will highlight the notes which are playing in red color.

![Image of MIDI player interface]

*Figure 9: Playing MIDI file in Player Controller Scene*
Creating a New MIDI File and Composing a song

Suppose a user want to create a new MIDI File and compose a new song. The user taps on “Create a new MID file” option on the main menu. The app will change the scene to Creating Page View Controller Scene. The user enters the file name in the text field. Suppose the user types “test” and tap on “create” button.

Figure 10: Creating Page View Controller Scene
The app changes its scene to Player Controller Scene with an empty note list. To add a new note message, fill the information of the note message into the corresponding text fields: note, beat time and duration. Then tap on “Add a Note” button. Suppose the user adds 5 notes in the track 0 which are C4, D4, E4, F4, G4 with increasing beat times from 1.0 to 5.0 and their durations are 1.0. The text view will show the information of these 5 notes. Tap on “Play” button will make the app play “do re mi fa so”. Suppose the user want to add another track which will play notes in drum. The user taps “Add a Track” button. A new track was just added. The user taps on the track text field. A picker appears at the bottom of the screen. The user selects the track 1 by scrolling the picker to “Track 1”. The text view loads the note list of track 1 which is empty.

![Figure 11: changing track in Player Controller Scene](image)
The user taps on the instrument text field that is beside the track text field. A picker with a list of instruments appear at the bottom of the screen. The user selects “SteelDrum” from the picker and taps “change instrument” button. The track 1 changes its instrument to steel drum. The user adds a few notes to track 1 and taps on the “Play” button. The app starts to play both track 0 and track 1.

![Image of the app interface](image)

*Figure 12: selecting instrument in Player Controller Scene*
The user taps on “save” button to save the MIDI file. The user can open the saved MIDI file from “main menu -> saved”.

Figure 13: saved MIDI files
Rename a saved file

Suppose the user want to rename “test.mid” to “test1.mid”. The user taps on “rename a MIDI file” in the main menu and then taps “test.mid”. The app changes its scene to Rename Page Controller Scene. The user enters “test1.mid” and taps the “rename button”. The MIDI file “test.mid” changes its name to “test1.mid”.

Figure 14: Rename Page Controller Scene
Delete a saved file

Suppose the user want to delete a save file “test1.mid”. The user taps “delete a MIDI file” option in the main menu. The user selects “test1.mid”. A pop up message appears on the screen: Are you sure to delete test.mid? The user taps “Yes” option. “test1.mid” is deleted.

![Image: Figure 15: deleting a MIDI file](image-url)
Import MIDI Files

You can import MIDI files from other application.

Figure 16: import MIDI file from other application
Methodology

In this section, I will talk about how to develop this application.

Create a New Project

Xcode -> File -> New -> Project... -> Single View App -> Next

Fill in the project name, your organization name, organization identifier and select your account. Click Next.

Table View Controller Scene

Select main.storyboard from the project navigator. Search and drag Table View Controller from the object library at the bottom right corner of Xcode to the canvas. Drag the arrow-shaped Entry Point to the Table View Controller Scene.
The Table View Controller Scene is controlled by UITableViewController by default. However, we want to make a custom controller class for this scene in order to gain more controls. Therefore, we need to add a new swift class file.

To create a new swift file, select File -> New -> File… -> Swift File -> Next -> change “File.swift” to “TableViewController.swift” -> Create.

Select “TableViewController.swift” from the project navigator. We need to type a few codes in this file.

```swift
import UIKit

class TableViewController: UITableViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```
Now we can change the controller of the Table View Controller Scene.

Select the Table View Controller Scene in Main.storyboard. Under the identity inspector section, change the class to TableViewController.

Figure 19: changing the controller of Table View Controller Scene to Table View Controller

Then we need to create several modes that menu may switch to:

```swift
enum TableMode: Int {
    case main, samples, saved, create, rename, delete
}
```

We need to set the first page of main menu:

```swift
var mainMenuCellList = ["Samples","Saved MIDI Files", "Create a new MIDI file", "rename a MIDI file", "delete a MIDI file"]
```
Then we need to make the table load these data and switch the menu’s mode. To do that, we need to override few functions of UITableViewController.

```swift
override func numberOfSections(in tableView: UITableView) -> Int {
    return 1
}

override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return 0
}

override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath)
    return cell
}

override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath){
}
```

The first function makes the number of sections of the table to 1. The second function will return the number of rows of the table. The third function will return a UITableViewCell for each row. We can use this function to make the table load the data. We need to change the identifier for the table view cell to “cell” under its attribute inspector section, since we want to reuse that table view cell for loading data for each row. The fourth function will be implemented when users select a row. We can use switch statement to switch the menu to different mode. After switching the mode, we can reload the table data by `tableView.reloadData()`.

For loading the sample MIDI file list or loading saved MIDI file list, you can create FileManager and then using FileManager.enumerator(at:includingPropertiesForKeys:options:errorHandler:) to search all MIDI file at a specific path.
func getSampleList() -> Array<String> {
    var sampleList = [String]()

    let folderURL = URL(fileURLWithPath: Bundle.main.bundlePath, isDirectory: true)
    let fileManager = FileManager.default
    let keys = [NSURLResourceKey.isDirectoryKey, NSURLResourceKey.localizedNameKey]
    let options: FileManager.DirectoryEnumerationOptions = [.skipsPackageDescendants, .skipsSubdirectoryDescendants, .skipsHiddenFiles]
    let enumerator = fileManager.enumerator(
        at: folderURL,
        includingPropertiesForKeys: keys,
        options: options,
        errorHandler: {(url, error) -> Bool in
            return true
        })

    if enumerator != nil {
        while let file = enumerator!.nextObject() {
            let pathURL = URL(fileURLWithPath: (file as! URL).absoluteString, relativeTo: folderURL)
            if pathURL.path.hasSuffix(".mid"){
                sampleList.append(pathURL.pathComponents.last!)
            }
        }
    }
    return sampleList
}

func getSavedList() -> Array<String> {
    var savedList = [String]()

    let documentsDirectory = NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).first!
    let folderURL = URL(fileURLWithPath: documentsDirectory, isDirectory: true)
    let fileManager = FileManager.default
    let keys = [NSURLResourceKey.isDirectoryKey, NSURLResourceKey.localizedNameKey]
    let options: FileManager.DirectoryEnumerationOptions = [.skipsPackageDescendants, .skipsSubdirectoryDescendants, .skipsHiddenFiles]
    let enumerator = fileManager.enumerator(
        at: folderURL,
        includingPropertiesForKeys: keys,
        options: options,
        errorHandler: {(url, error) -> Bool in
            return true
        })

    if enumerator != nil {
        while let file = enumerator!.nextObject() {
            let pathURL = URL(fileURLWithPath: (file as! URL).absoluteString, relativeTo: folderURL)
            if pathURL.path.hasSuffix(".mid"){
                savedList.append(pathURL.pathComponents.last!)
            }
        }
    }
    return savedList
}
I put those two global functions in the “utilities.swift”.

**Changing Scenes**

We can change scenes by using segues. To create a segue, ctrl + click the scene you want to perform and drag to the destination scene. A segue menu will appear. Select “Present Modally”.

![Figure 20: adding a segue](image)

After successfully adding a segue, an arrow will connect both scenes.
Figure 21: segue

To perform segue, we can use `performSegue(withIdentifer:, sender:)`.

To send data or message through segue, we need to override the function `prepare(for segue:, sender:)`.

```swift
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if segue.identifier == "segue1" {
        let seguel = segue.destination as! UIViewController
        seguel.anyvalue = value
    }
}
```
Player Controller View Scene

The Player Controller Scene contains these functions: play the MIDI file, add a Track, delete a track, change instrument, add a note, delete a note and save. I will talk about each of these functions in this section.

Figure 22: MIDI Player Scene
Play and Reset

To create a Play button, search and drag the Button from the object library to the canvas.

Double click on the button and change the button’s label text to “Play”. We need to create an action function for this button. This action will be implemented every time the button is clicked.

Change the editor to Assistance Editor by click icon at the top right corner. Select the left canvas to be storyboard and select the right canvas to be MIDIPlayerController.swift. Ctrl + drag from the Player button to the right swift file. A menu appears. Select action for Connection.

Enter the action function’s name. select UIButton for Type.

Figure 23: create an action function for a button
After successfully creating an action function, the action function will be automatically added
into the swift file.

```swift
@IBAction func play(_ sender: UIButton) {
}
```

In order to play the MIDI file, we need to extract the data from the MIDI file. We need to import
the AudioToolBox framework to do this. This framework provides interfaces for recording or
playing audio, converting formats, parsing audio streams and configuring audio session. (Apple
Inc., 2017b)

To load data from MIDI files, we can use `MusicSequenceFileLoad(MusicSequence, CFURL,
MusicSequenceFileTypeID, MusicSequenceLoadFlags)`. We need to create a `MusicSequence`
variable to store the loaded data. To create a new `MusicSequence`, we need to use

`NewMusicSequence(UnsafeMutablePointer<MusicSequence?>)`.

```swift
func getSequenceFromFile(name: String) -> MusicSequence {
    var musicSequence: MusicSequence?
    NewMusicSequence(&musicSequence)

    var name = name
    if !name.hasSuffix(".mid") {
        name.append(".mid")
    }

    let documentsDirectory = NSSearchPathForDirectoriesInDomains(.documentDirectory,
        .userDomainMask, true).first!

    if let fileURL = URL(fileURLWithPath: documentsDirectory).appendingPathComponent(name) {
        // print(fileURL)
        MusicSequenceFileLoad(musicSequence!,
            fileURL as CFURL,
            .midiType,
            .smf_ChannelsToTracks)
    }
    return musicSequence!
}
```
After loading the MIDI data, we need to set up a player to play the MIDI data. AVFoundation framework provides a nice MIDI player interface which is AVMIDIPlayer. (Apple Inc., 2017a)

To initialize a new AVMIDIPlayer we need to convert our MusicSequence to Data structure and we need a sound bank and we need to get the time resolution from our MusicSequence.

For converting MusicSequence to Data structure, we can use function

MusicSequenceFileCreateData(MusicSequence, MusicSequenceFileTypeID, MusicSequenceFileFlags, Int16, UnsafeMutablePointer<Unmanaged<CFData>?>). Don’t forget to release the data after you use it.

For the sound bank, I used the NTONYX’s SoundFont 32Mb GM Stereo Set. (NTONYX, 2017)

For getting time resolution, firstly, we can get the tempo track from the MusicSequence by

MusicSequenceGetTempoTrack(MusicSequence, UnsafeMutablePointer<MusicTrack?>).

Then we can use the function MusicTrackGetProperty(MusicTrack, UInt32, UnsafeMutableRawPointer, UnsafeMutablePointer<UInt32>) to get the time resolution.
func determineTimeResolution(musicSequence: MusicSequence) -> UInt32 {
    var track: MusicTrack?
    var status = MusicSequenceGetTempoTrack(musicSequence, &track)

    if let tempoTrack = track {
        var propertyLength = UInt32(0)
        var timeResolution = UInt32(0)
        status = MusicTrackGetProperty(tempoTrack, kSequenceTrackProperty_TimeResolution, &timeResolution, &propertyLength)

        if status != noErr {
            CheckError(status)
        }
    }

    return timeResolution
} else {
    print("error getting tempo track")
    return 0
}

func createAVMIDIPlayer(musicSequence: MusicSequence) {

    // http://www.nTonyx.com/sf_f.htm
    let bankURL = Bundle.main.url(forResource: "32MbgMSTereo", withExtension: "sf2")
    var status = noErr
    var data: Unmanaged<CFData>?

    timeResolution = determineTimeResolution(musicSequence: musicSequence)
    status = MusicSequenceFileCreateData(musicSequence, .midiType, .eraseFile, Int16(timeResolution), &data)

    if status != noErr {
        CheckError(status)
    }

    if let md = data {
        let midiData = md.takeUnretainedValue() as Data
        do {
            try self.avMIDIPlayer = AVMIDIPlayer(data: midiData as Data, soundBankURL: bankURL)
        } catch let error as NSError {
            print("nil midi player")
            print("Error \(error.localizedDescription)")
        }
        data?.release()
        self.avMIDIPlayer?.prepareToPlay()
    }
}
After call AVMIDIPlayer.prepareToPlay(), you are ready to play the AVMIDIPlayer. You can use AVMIDIPlayer.play(AVMIDIPlayerCompletionHandler? = nil) to play it and AVMIDIPlayer.stop() to stop it.

@IBAction func play(_ sender: UIButton) {
    createAVMIDIPlayer(musicSequence: musicSequence!)
    if (avMIDIPlayer?.isPlaying)! {
        avMIDIPlayer?.stop()
        sender.setTitle("Play", for: .normal)
    } else {
        avMIDIPlayer?.play(nil)
        sender.setTitle("Stop", for: .normal)
    }
}

AVMIDIPlayer.currentPosition is the variable that represent the current playback position in seconds. Setting it to 0 can play the song from the beginning.

@IBAction func reset(_ sender: UIButton) {
    avMIDIPlayer?.currentPosition = 0
}

To show and update AVMIDIPlayer.currentPosition on a label while the AVMIDIPlayer is playing, we can use Timer.

    func startTimer () {
        if timer == nil {
            timer = Timer.scheduledTimer(timeInterval: 0.01, target: self,
                                          selector: #selector(tick),
                                          userInfo: nil, repeats: true)
        }
    }

    func stopTimer() {
        if timer != nil {
            timer.invalidate()
        }
    }

    @objc func tick (){
        if avMIDIPlayer!.isPlaying {
            timeTextField.text = "\((avMIDIPlayer!.currentPosition*100).rounded(.up)/100)"
        }
    }
Show the List of Notes in a Text View

Figure 24: showing list of note in the text view
The Text View can be searched from the Object Library and dragged on the Scene. You can create its connection to swift file by the method of ctrl + drag that is similar to creating an action function.

To show the note list of the selected track, we need to extract notes from that track. We can use `MusicEventIteratorGetEventInfo(MusicEventIterator, UnsafeMutablePointer<MusicTimeStamp>, UnsafeMutablePointer<MusicEventType>, UnsafeMutablePointer<UnsafeRawPointer?>, UnsafeMutablePointer<UInt32>)` to get the note information. To do that, first we need to create a new MusicEventIterator by `NewMusicEventIterator(MusicTrack, UnsafeMutablePointer<MusicEventIterator?>)`. Then, we can loop in a while statement by using `MusicEventIteratorNextEvent(MusicEventIterator)` and `MusicEventIteratorHasCurrentEvent(MusicEventIterator, UnsafeMutablePointer<DarwinBoolean>)`

```swift
func getNoteListFromMusicTrack(musicTrack: MusicTrack) -> Array<Note> {
    var noteList = Array<Note>()
    var hasCurrentEvent: DarwinBoolean = DarwinBoolean.init(true)
    var musicEventIterator: MusicEventIterator?
    var musicEventType: MusicEventType = 0
    var musicTimeStamp: MusicTimeStamp = 0.0
    var musicEventDataRef: UnsafeRawPointer?
    var eventDataSize: UInt32 = 0
    NewMusicEventIterator(musicTrack, &musicEventIterator)
    MusicEventIteratorHasCurrentEvent(musicEventIterator!, &hasCurrentEvent);
    while (hasCurrentEvent).boolValue {
        MusicEventIteratorGetEventInfo(musicEventIterator!, &musicTimeStamp,
                                         &musicEventType, &musicEventDataRef, &eventDataSize)
        if musicEventType == kMusicEventType_MIDINoteMessage {
            let eventData = musicEventDataRef!.load(as: MIDINoteMessage.self)
            let note = Note(noteInfo: eventData, time: musicTimeStamp)
            noteList.append(note)
        }
        MusicEventIteratorNextEvent (musicEventIterator!);
        MusicEventIteratorHasCurrentEvent (musicEventIterator!, &hasCurrentEvent);
    }
    return noteList
}
```
We can use UITextView.insertText(String) to update the table view.

```swift
func updateTextView () {
    textView.text = ""
    textView.insertText(textViewTitle)

    for (index,note) in noteList.enumerated() {
        var indexString = "\(index)".prefix(5)
        indexString += String(repeating: " ", count: (5 - indexString.count))

        var noteString = NoteString[Int(note.noteInfo.note)].prefix(5)
        noteString += String(repeating: " ", count: (5 - noteString.count))

        var channelString = "\(note.noteInfo.channel)".prefix(7)
        channelString += String(repeating: " ", count: (7 - channelString.count))

        var timeString = "\(convertBeatToTime(inSequence: musicSequence!,inBeat: note.time!))".prefix(5)
        timeString += String(repeating: " ", count: (5 - timeString.count))

        var beatString = "\(note.time!)".prefix(5)
        beatString += String(repeating: " ", count: (5 - beatString.count))

        var durationString = "\(note.noteInfo.duration)".prefix(7)
        durationString += String(repeating: " ", count: (7 - durationString.count))

        textView.insertText("\\(indexString) \(noteString) \(channelString) \(timeString) \(beatString) \(durationString) \n")
    }
}
```

To highlight the playing notes in the Text View, we can create an NSMutableAttributedString and set UITextView.attributedText to this string.

```swift
let attributedString = NSMutableAttributedString(string: textView.text)
attributedString.addAttribute(NSAttributedStringKey.backgroundColor, value: UIColor.red, range: NSRange(location: 0, length: attributedString.length))
textView.attributedText = attributedString
textView.font = UIFont(name: "Courier new", size: 12)
```

We can use UITextView.scrollRangeToVisible( NSRange) to automatically scroll the text view to the highlighted note position.
Change a track’s instrument

First, we need to get the information of all instruments in your sound bank. Instrument information includes MSB (Most Significant Bit), LSB (Least significant bit), program number, instrument name. That information is packed into a CFArray which can be type casted to NSArray. After printing CFArray, I found that this array was actually a collection of NSDictionary. Therefore, we can loop into this array, and extract the information from those NSDictionary.

```swift
func getInstrumentList () -> Array<Instrument> {  
    var instrumentList = Array<Instrument> ()  
    let bankURL = Bundle.main.url(forResource: "32MGMStereo", withExtension: "sf2")  
    var instrumentsInfo:Unmanaged<CFArray>?  
    CopyInstrumentInfoFromSoundBank(bankURL! as CFURL, &instrumentsInfo)  
    let instrInfo = instrumentsInfo!.takeRetainedValue() as! NSArray  
    for i in instrInfo {  
        let lsb = (i as! NSDictionary).value(forKey: "LSB") as! Int  
        let msb = (i as! NSDictionary).value(forKey: "MSB") as! Int  
        let name = (i as! NSDictionary).value(forKey: "name") as! String  
        let program = (i as! NSDictionary).value(forKey: "program") as! Int  
        let instrument = Instrument(LSB: lsb, MSB: msb, name: name, program: program)  
        instrumentList.append(instrument)  
    }  
    instrumentList.sort(by: {(a:Instrument,b:Instrument) in  
        return a.name < b.name  
    })  
    return instrumentList  
}
```

Once we get the list of the instruments’ information, we can create a UIPickerView which allows the users to pick the instrument they want. Let MIDIPlayerController inherit from UIPickerViewDelegate and UIPickerViewDataSource. Create an instrument picker view:

```swift
instrumentPickerView = UIPickerView(frame: CGRect(x: screenSize.minX, y: screenSize.minY, width: screenSize.width, height: screenSize.height/3))  
instrumentPickerView.delegate = self  
instrumentPickerView.dataSource = self  
instrumentTextField.inputView = instrumentPickerView  
instrumentPickerView.selectRow(0, inComponent: 0, animated: false)  
instrumentTextField.text = instrumentList[0].name
```
And implement functions from UIPickerViewDataSource:

```swift
func numberOfComponents(in pickerView: UIPickerView) -> Int {
    return 1
}

func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
    if pickerView == instrumentPickerView {
        return instrumentList.count
    }
    return 0
}

func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int) -> String? {
    if pickerView == instrumentPickerView {
        return instrumentList[row].name
    }
    return"
}

func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
    if pickerView == instrumentPickerView {
        instrumentTextField.text = instrumentList[row].name
    }
}
```

Reload the picker’s data by UIPickerView.reloadAllComponents().

We can change the instrument of a track by MusicTrackNewMIDIChannelEvent(MusicTrack, MusicTimeStamp, UnsafePointer<MIDIChannelMessage>). To change instrument, we need to add MIDIChannelMessage two times. In the first one, we need to set the pitch bend which includes MSB and LSB. In the second one, we need to set the program. When we set the MIDIChannelMessage’s status to 0xE0, it means this MIDIChannelMessage will set MSB to data1 and LSB to data2. When we set the MIDIChannelMessage’s status to 0xC0, it means this MIDIChannelMessage will set the program number to data1. (Marshall, 2001)
Save the MIDI File

We can save the MIDI file by `MusicSequenceFileCreate(MusicSequence, CFURL, MusicSequenceFileTypeID, MusicSequenceFileFlags, Int16)`.

```swift
@IBAction func changeInstrument(_ sender: UIButton) {
    if instrumentTextField.text == nil { return }
    let index = instrumentPickerView.selectedRow(inComponent: 0)
    let instrument = instrumentList[index]

    var inMessage = MIDIClaimMessage(status: 0xE0, data1: UInt8(instrument.MSB), data2: UInt8(instrument.LSB), reserved: 0)
    MusicTrackNewMIDIClaimEvent(musicTrack!, 0, &inMessage)

    inMessage = MIDIClaimMessage(status: 0xC0, data1: UInt8(instrument.program), data2: 0, reserved: 0)
    MusicTrackNewMIDIClaimEvent(musicTrack!, 0, &inMessage)

    musicSequenceModifiedFlag = true
}
```

```swift
func createMIDIFile(sequence:MusicSequence, fileName: String) {
    let documentsDirectory = 
    NSSearchPathForDirectoriesInDomains(.documentDirectory, .userDomainMask, true).first!

    var name = fileName
    if !name.hasSuffix(".mid"){
        name.append(".mid")
    }

    if let fileURL = NSURL(fileURLWithPath: 
    documentsDirectory).appendingPathComponent(name) {
        let timeResolution = determineTimeResolution(musicSequence: sequence)
        let status = MusicSequenceFileCreate(sequence, fileURL as CFURL, .midiType,
        [.eraseFile], Int16(timeResolution))
        if status == noErr {
            CheckError(status)
        }
    }
}
```
Delete a Track

We can use `MusicSequenceDisposeTrack(MusicSequence, MusicTrack)` to dispose a track.

Delete a Note

`MusicTrackClear(MusicTrack, MusicTimeStamp, MusicTimeStamp)` can be used to delete all notes during a period of time. But if we only want to delete one note, before we clear the track, we need to store the notes in that period except the one we want to delete. After we call the `MusicTrackClear` function we can insert those affected notes back to the track.

```swift
func deleteANoteAt(index: Int) {
    if index >= noteList.count {
        return
    }
    let beatTime = noteList[index].time!
    var tempNoteList = Array<Note>()
    for (i, note) in noteList.enumerated() {
        if note.time >= beatTime && note.time < beatTime + 0.1 && i != index {
            tempNoteList.append(note)
        } else if note.time > beatTime + 0.1 {
            break
        }
    }
    MusicTrackClear(musicTrack!, beatTime, beatTime + 0.1)
    for note in tempNoteList {
        insertANote(note: note.noteInfo.note,
                    time: note.time,
                    channel: note.noteInfo.channel,
                    velocity: note.noteInfo.velocity,
                    releaseVelocity: note.noteInfo.releaseVelocity,
                    duration: note.noteInfo.duration)
    }
    noteList = getNoteListFromMusicTrack(musicTrack: musicTrack!)
}
```
Import MIDI Files

You need to register a Document Type and an Imported UTIs in the Info.plist file.

Figure 25: register a Document Type and an Imported UTIs in the Info.plist file

You can receive the file from the function `application(_ application: UIApplication, willFinishLaunchingWithOptions launchOptions:?[UIApplicationLaunchOptionsKey: Any]?) -> Bool` and `application(_ app: UIApplication, open url: URL, options: [UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool`. Both functions should be written in `AppDelegate.swift`. 
func application(_ application: UIApplication, willFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    if launchOptions == nil {
        return true
    }

    for option in launchOptions! {
        switch option.key {
        case UIApplicationLaunchOptionsKey.url:
            saveFileFromURL(url: option.value as! URL)
            break
        case UIApplicationLaunchOptionsKey.sourceApplication:
            break
        case UIApplicationLaunchOptionsKey.annotation:
            break
        default:
            break
        }
    }

    return true
}

func application(_ app: UIApplication, open url: URL, options: [UIApplicationOpenURLOptionsKey : Any] = [:]) -> Bool {
    saveFileFromURL(url: url)
    return true
}
Results

Most features I promised in my proposal has been achieved in MIDIPlayer except for one. The feature I failed to realize is to convert MIDI files into Music Sheet. Other than that, MIDIPlayer can import and play MIDI file with pretty nice sound. Thanks to the sound bank from NTONYX. User can do some simple modification on MIDI files and they can even create a new MIDI file and compose new songs. I thank this application is a very useful tool for people who want to play or edit MIDI file on their iPhone.

Improvement in the Future

I can continue on realizing the feature that convert MIDI file to sheet music. The best way to do this is to convert MIDI file to MusicXML first. Then visualize MusicXML to sheet music. I can also add more controls for the MIDIPlayer. For example, adjusting tempo, adjusting volume.

I can also add more features to MIDIPlayer, for instance, metronome.
Bibliography


