U S E R   G U I D E

# SIMPLE MIDI 1.0

UNITY3D 3.5.6

Eduardo Calçada — Revision 1

# MIDI Connection

## Setting up Plugin

- Be sure to include the MIDIPlugin.dll file inside a folder called “Plugins” within your game root folder.

## Loading the Plugin inside Unity

- Before we can use the plugin’s features we must first declare the following functions, as can be seen on the example project. These are placed inside the **class scope**.

```
// Plugin function imports

        [DllImport ("MIDIPlugin")]
         private static extern int GetMIDIDevices();

        [DllImport ("MIDIPlugin")]
         private static extern bool SelectMIDIDevice(int id);

        [DllImport ("MIDIPlugin")]
         private static extern bool OpenMIDIPort();

        [DllImport ("MIDIPlugin")]
         private static extern void StartListenMIDI();

        [DllImport ("MIDIPlugin")]
         private static extern int GetMIDIMessageVelocity();

        [DllImport ("MIDIPlugin")]
         private static extern int GetMIDIMessageKey();

        [DllImport ("MIDIPlugin")]
        private static extern bool CloseMIDI();
```

## Connecting to a MIDI Device

- Prior to the actual connection process, it’s wise to make sure the computer detects whether there is a **MIDI device** plugged in. This is done through the “GetMIDIDevices()” function which **returns** the number of devices connected ranging from **0**, onwards. If there are no devices, the function **returns a -1**.

- If the previous stage is successful, we then want to let the plugin know which device we want to connect to. This is done through the "SelectMIDIDevice(int id)" function and it simply asks for the **device ID**. If there is only one device **connected**, then a **0** is **passed in.** If there are **multiple devices**, then a we might want to pass in a **1 for instance**. The function returns whether the **operation** was **successful** or not. This selection process is more akin to trial and error given the current build. (A later update will be introduced to streamline this process)

## Opening a MIDI Port & Listening

- After we selected the **correct device**, the next step would be to open a **MIDI port**. This is done through the "OpenMIDIPort()" function and it returns a **boolean**.

- Now that the port is opened, we simply need to call the "StartListenMIDI()" method and the plugin will start **running in the background** to **collect the MIDI messages**. Also returns a **boolean**.

- The code excerpt below (from the example project) elaborates on these concepts:

```
Debug.Log ("There are these many input devices: " + GetMIDIDevices().ToString());

        // Selecting the MIDI Keyboard (0 since it is the only MIDI Device connected)
        if(!SelectMIDIDevice(0))
                return false;

        // Setting up the MIDI connection
        if(!OpenMIDIPort())
                return false;

        // Now listening for any incoming MIDI Messages
        StartListenMIDI();
```

## <u>Retrieving MIDI Keys & Velocities</u>

- Now that the plugin is ready to be accessed, in order to get the **most recent pressed key** and it's **respective velocity** we call the following functions:

```
// Storing the most recent MIDI messages
int keyNumber   = GetMIDIMessageKey();
int keyVelocity = GetMIDIMessageVelocity();
```

- As can be seen, the functions above call the plugin and retrieve the **key information**; then storing it in a simply integer. However, calling them on the Unity3D **main-thread loop** will prove to be rather slow for detecting more than **3 key presses** at a time. To solve this, it is suggested that this functionality be included within a class and then given it's own thread to run on. This will **significantly improve response times** and accuracy. (The example provided runs the MIDI input on a separate thread with full documentation)

## <u>Closing MIDI</u>

- After the game/application **no longer requires** the services of the MIDI Plugin, we can simply close it with the "CloseMIDI()" function. This will **automatically shut down** the used MIDI port and will stop running the plugin.

## <u>MIDI Example</u>

- Included in this package is a fully working MIDI set-up. With any MIDI keyboard it should be possible to determine which keys are being pressed and at what velocities.

## <u>Further Information</u>

- If you have any questions or concerns/suggestions then please don't hesitate to contact me at eduardocalcada@gmail.com. I hope you enjoy using this plugin.