Java, RtMidi und USB-MIDI-Geräte

im März 2024

von Friedrich Forck

Inhaltsverzeichnis

Einleitung

Java besitzt seit seiner Version 1.3 auch ein Midi-Package: javax.sound.midi. Bis zur Version 1.8 (auch Version 8 genannt) funktionierten damit erstellte Programme auch dann, wenn USB-Midi-Geräte verwendet wurden. Mit der Version 9 hörte das auf. Dass der Bug existiert ist bekannt. Er wurde am 25.10.2017 gemeldet und ist bis heute (März 2024) nicht behoben.[1]

Mit dieser Arbeit zeige ich einen Weg auf, wie man trotzdem mit Java 22 ein Midi-Programm erstellen kann, welches USB-Geräte erkennt und die Ein- und Ausgaben richtig behandelt. Ich verwende dabei Methoden aus dem Package foreign. Siehe dazu das Kapitel Foreign Function & Memory API.
Um die Schwierigkeiten mit dem USB-Anschluss von Midi-Geräten zu umgehen, werden für den Midi-Input und Output mit C++ geschriebene Methoden verwendet. Dazu wird mit der Open-Source-Bibliothek RtMidi in der Entwicklungsumgebung Visual-Studio (Community) eine DLL erstellt, auf die dann aus dem Java-Programm heraus zugegriffen wird. Die Plattform ist Windows. In ähnlicher Weise muss es auch möglich sein unter Macintosh OS X oder Linux vorzugehen. Denn wie Java ist auch die Bibliothek RtMidi für alle drei Plattformen geschrieben.

RtMidi

RtMidi ist ein Satz von C++-Klassen (RtMidiIn, RtMidiOut und API-spezifische Klassen), die eine gemeinsame API für Echtzeit-MIDI-Ein- und Ausgabe bereitstellen. Die unterstützten Betriebssysteme sind Linux (ALSA & JACK), Macintosh OS X (CoreMIDI & JACK), und Windows (Multimedia Library). RtMidi wurde mit folgenden Zielen konzipiert:

- objektorientiertes C++-Design
- einfache, gemeinsame API für alle unterstützten Plattformen
- nur ein Header und eine Quelldatei zur einfachen Einbindung in Programmierprojekte
- die Aufzählung von MIDI-Geräten soll möglich sein

Midi-Klassen in Java

Die Klassen für die Midi-Programmierung mit Java befinden sich im Package javax.sound.midi. Die Klasse MidiSystem umfasst das ganze MidiSystem auf dem Computer. Sie hat z.B. die Methode getMidiDeviceInfo(), die ein Array mit Informationen über alle vorhandenen Midi-Geräte zurück gibt. Mit Klassen, die das Interface MidiDevice

1 https://bugs.openjdk.java.net/browse/JDK-8190455

implementieren, kann man alle Midi-Geräte repräsentieren, einschließlich einem Sequenzer oder einem Synthesizer. Instanzen von Klassen vom Typ des Interfaces Transmitter senden MidiEvent-Objekte an Instanzen von Klassen des Typs Receiver. Ein Receiver muss die Methode send(MidiMessage message, long timestamp) überschreiben. Mit dieser Methode werden Midi-Informationen an ein MidiOut oder ein Objekt der Klasse Synthesizer gesendet.
Weitere Informationen zum Midi-Package sind im Internet leicht zu finden.

Der Weg, Midi-Daten von einem an den Computer angeschlossenen Keyboard an ein MidiOut zu senden ist:
1. MidiDevice implementierende Klassen vom MidiIn und MidiOut erstellen.
2. Den Transmitter der Klasse vom MidiIn ermitteln und den Receiver der Klasse vom MidiOut.
3. Zuletzt wird beim gefundenen Transmitter mit setReceiver() der gefundenen Receiver gesetzt.

Da ein an den Computer angeschlossenes Keyboard jedoch einen USB-Port benutzen wird, ist diese Vorgehensweise seit Java 9 nicht mehr möglich.

Foreign Function & Memory API

Diese seit Java 17 als Preview Feature zur Verfügung stehende Funktionalität wird auf der Webseite https://openjdk.org/jeps/454 beschrieben. Sie ist seit der Version 22 kein Preview Feature mehr. Die mit den Versionen 18 und 19 eingeführten Änderungen waren noch recht erheblich. Nun sind die endgültigen Klassen- und Methodennamen gefunden und die Methodensignaturen festgelegt.
Mit den im Package java.lang.foreign zur Verfügung gestellten Klassen ist es möglich, auf C-Funktionen der Betriebssysteme oder von DLLs und deren Äquivalenten unter Linux und Macintosh OS X zuzugreifen.

Ein kleines USB-taugliches Java-Programm

Um mit Java Programme zu schreiben, die auch mit USB-Geräten umgehen können, ohne die Version 8 oder niedriger benutzen zu müssen, schlage ich den im Folgenden beschriebenen Weg vor. Benötigt werden dazu die Entwicklungsumgebungen Visual Studio 2019 (Community) und Eclipse. Außerdem die C++-Bibliothek RtMidi und die Java-Version 18 (oder höher).

RtMidi unter Windows verwenden

Zuerst wird die Bibliothek von der Webseite https://github.com/thestk/rtmidi heruntergeladen. (s. Abbildung 1). Die erhaltene ZIP-Datei rtmidi-master.zip wird an einem geeigneten Ort entzippt.

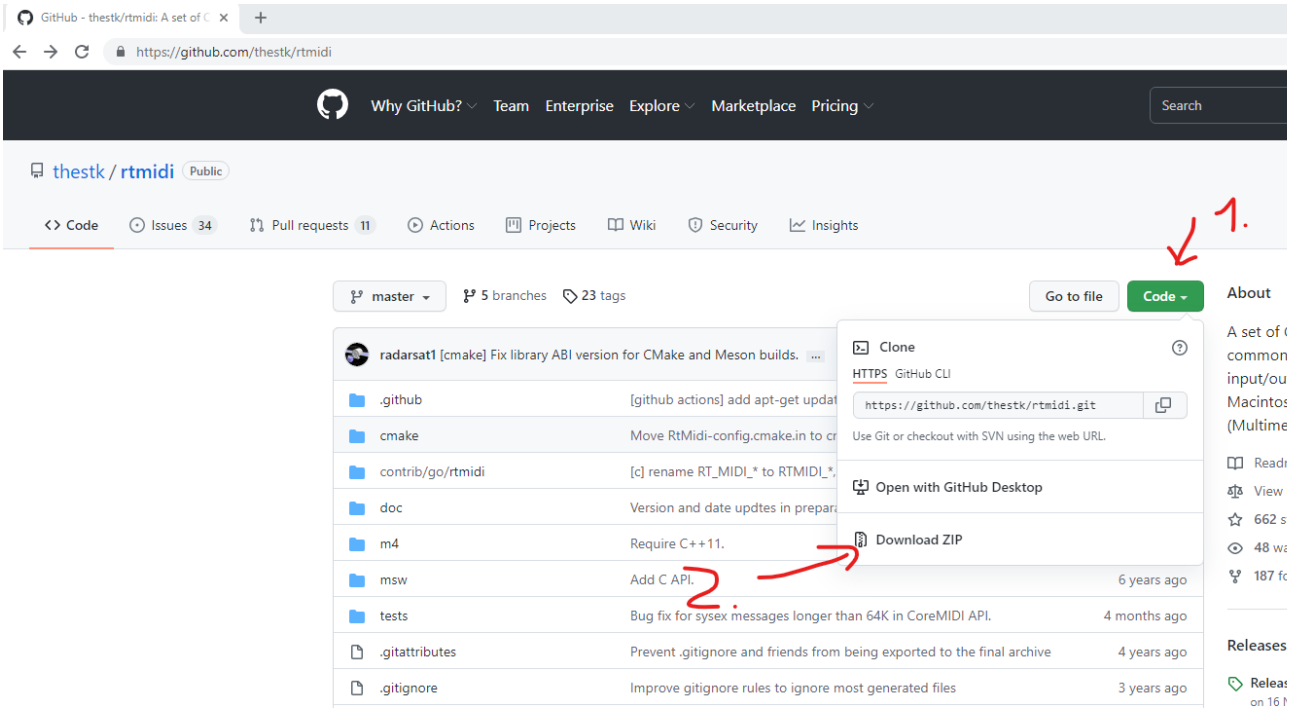
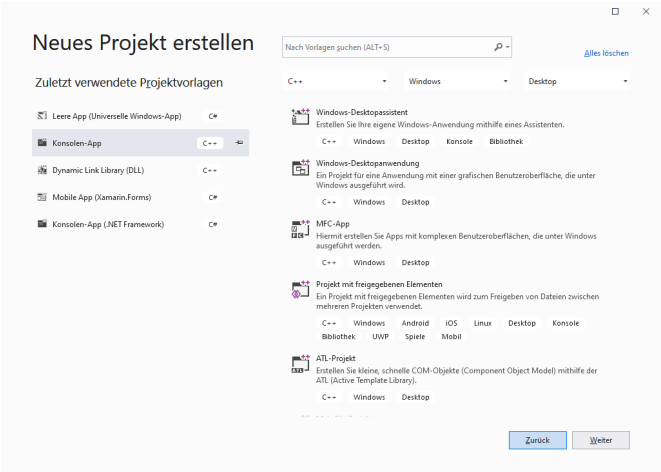


Abbildung 1: Download von RtMidi

Aus dem Inhalt sind hier die Dateien RtMidi.h und RtMidi.cpp sowie der Ordner tests wichtig.

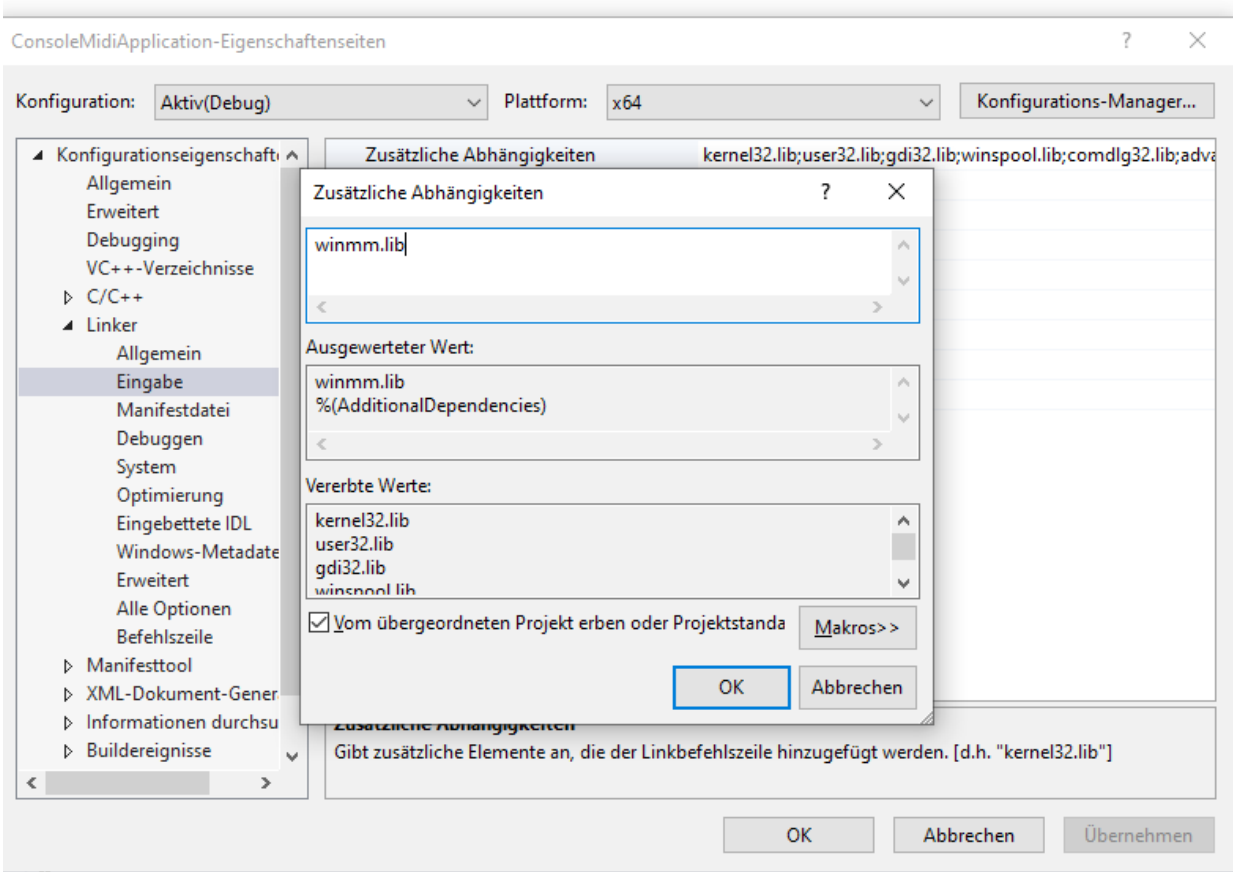
Nun wird RtMidi überhaupt einmal unter Windows verwendet:

In Visual Studio erstellen wir ein neues Projekt. Als Typ wird „Konsolen-App" gewählt. Ich schlage als Projekt-Namen „ConsoleMidiApplication" vor.



Zwei weitere Schritte sind nun noch notwendig:

Zuerst wird unter dem Menüpunkt „Projekt“ der Punkt „ConsoleMidiApplication-Eigenschaften“ ausgewählt. Unter Linker → Eingabe → Zusätzliche Abhängigkeiten wird „winmm.lib“ eingegeben.



Dann wird die Datei Rtmidi.h oben um die Zeilen

```
#ifndef __WINDOWS_MM__
#define __WINDOWS_MM__
#endif
```

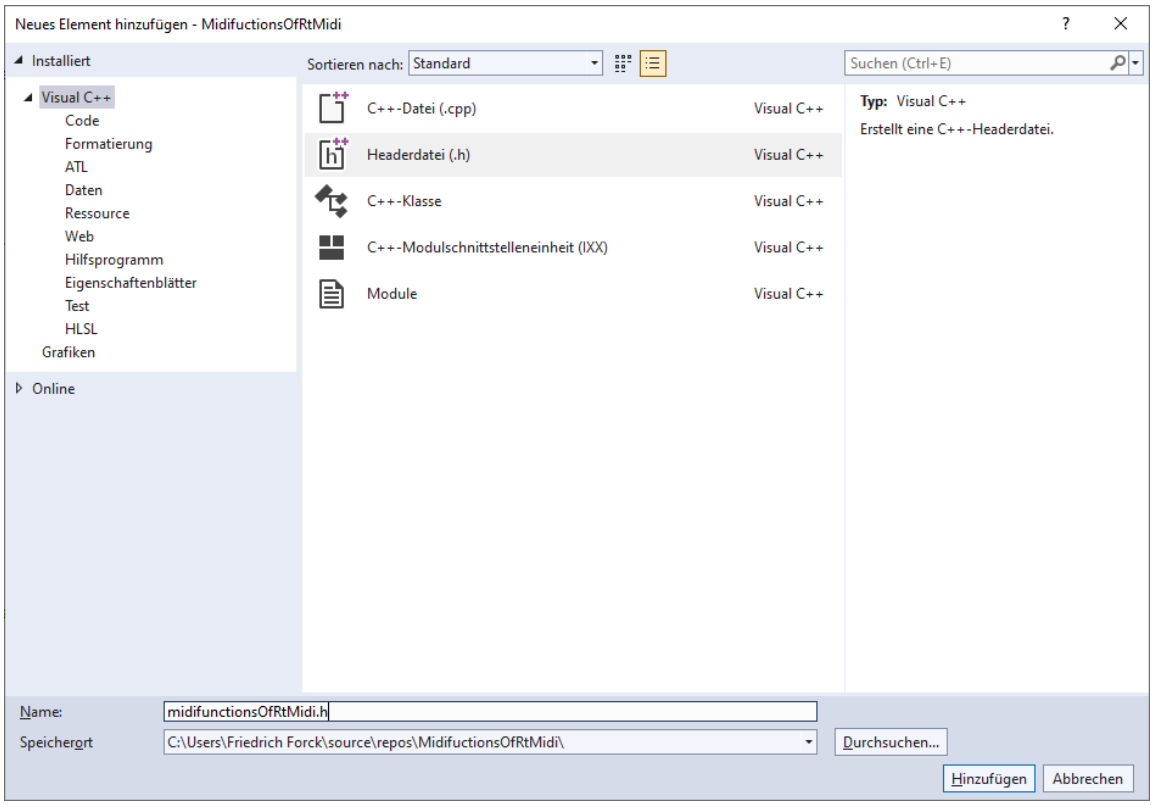
ergänzt.

Diese Schritte werden im RtMidi-Tutorial (https://www.music.mcgill.ca/~gary/rtmidi/) unter dem Punkt Compiling genannt.

Jetzt können die verschiedenen Quellcodes aus den .cpp-Dateien im heruntergeladenen Ordner tests in die Quelldatei ConsoleMidiApplication.cpp kopiert werden und so kannst du nun sehen, dass du RtMidi auf deinem Rechner zum Laufen gebracht hast. Der Inhalt von midiout.cpp (von Gary Scavone, 2003-2004) z.B. führt dazu, dass ein Ton erklingt und dass im Konsolenfenster die verfügbaren MidiOut-Ports angezeigt werden.

Midi-Funktionen aus RtMidi als DLL

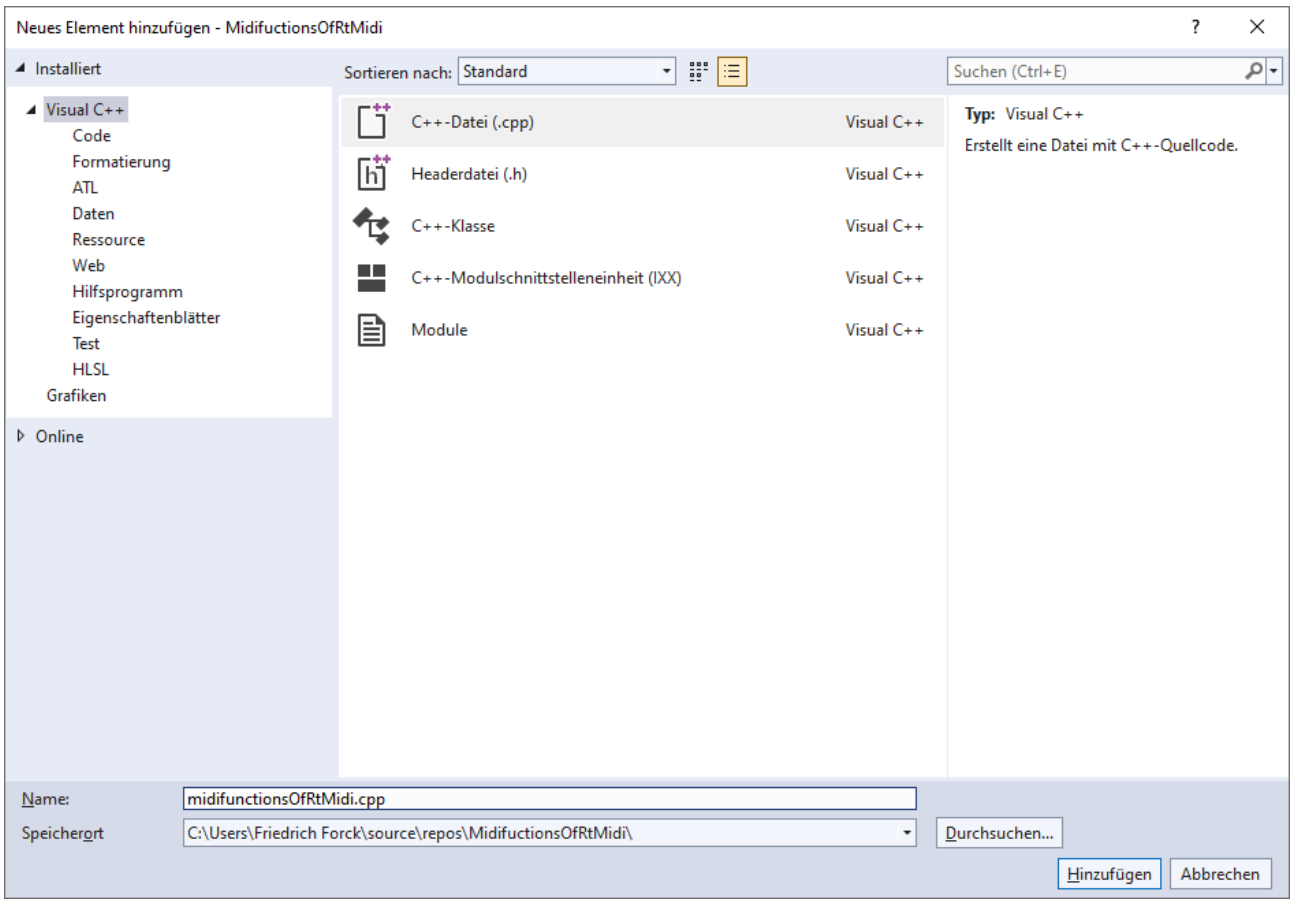
Es wird in Visual Studio ein neues Projekt erstellt. Als Projektvorlage nehmen wir Dynamic Link Library (DLL) und vergeben den Namen MidifunctionsOfRtMidi. Automatisch generiert sind jetzt die Headerdateien framework.h und pch.h sowie die Quelldateien dllmain.cpp und pch.cpp.
Im Projektmappen-Explorer rufen wir über dem Eintrag „Headerdateien" das Kontextmenü auf und wählen Hinzufügen → Neues Element. Dann wählen wir „Headerdatei" aus dem erschienenen Dialog aus und vergeben den Namen midifunctionsOfRtMidi.h.



In die Headerdatei kommt der Code

```
#ifdef __cplusplus
extern "C" {
#endif
	__declspec(dllexport) int receiveMidi(int portNum);
	__declspec(dllexport) int setPortOfMidiIn(int portNum);
	__declspec(dllexport) void sendMidiMessage(unsigned int statusAndChannel,
                                               unsigned int data1, unsigned int data2);
	__declspec(dllexport) int setPortOfMidiOut(int portNum);
#ifdef __cplusplus
}
#endif
```

Entsprechend wird nach einem Rechtsklick auf Quelldateien eine C++-Datei hinzufügt:



Der Code für die Datei MidifunctionsOfRtMidi.cpp lautet:

```
#include "pch.h"
#include <Windows.h>
#include <iostream>
#include <signal.h>
#include "RtMidi.h"
#include "midifunctionsOfRtMidi.h"


#define RETURN_SUCCESS 0
#define RETURN_FAILURE 1

bool done;
static void finish(int /*ignore*/) { done = true; }

static RtMidiIn* midiin = 0;
static unsigned int portOfMidiIn = 0;
static RtMidiOut* midiout;
static unsigned int portOfMidiOut = 0;
std::vector<unsigned char> message(3);

__declspec(dllexport) int receiveMidi(int portNum) {
    std::vector<unsigned char> message;

    int nBytes;

    try {
        if (midiin == 0)
            midiin = new RtMidiIn();
```

```
    }
    catch (RtMidiError) {
        return RETURN_FAILURE;
    }

    if (portNum >= midiin->getPortCount()) {
        delete midiin;
        return RETURN_FAILURE;
    }

    // ignore sysex, timing, active sensing messages.
    midiin->ignoreTypes(true, true, true);

    // Install an interrupt handler function.
    done = false;
    (void)signal(SIGINT, finish);

    // Periodically check input queue.
    while (!done) {
        midiin->getMessage(&message);
        nBytes = (int)message.size();
        for (int i = 0; i < nBytes; i++)
            if (nBytes > 0) {
                int midiMessage = (message[0] << 16) + (message[1] << 8) + message[2];
                return midiMessage;
            }
        Sleep(1);
    }
    return RETURN_SUCCESS;
}

__declspec(dllexport) int setPortOfMidiIn(int portNum){
    try {
        if (midiin == 0)
            midiin = new RtMidiIn();
    }
    catch (std::exception) {
        return RETURN_FAILURE;
    }

    try {
        if (portNum >= midiin->getPortCount()) {
            delete midiin;
            return RETURN_FAILURE;
        }
        if (portOfMidiIn != portNum) {
            midiin->closePort();
            portOfMidiIn = portNum;
        }
        if (!midiin->isPortOpen()) {
            midiin->openPort(portNum);
            if (!midiin->isPortOpen()) {
                //delete midiin;
                return RETURN_FAILURE;
            }
        }
        return RETURN_SUCCESS;
    }
    catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        delete midiin;
        return RETURN_FAILURE;
    }
```

```
}

__declspec(dllexport) int setPortOfMidiOut(int portNum){
    try {
        if (midiout == 0)
            midiout = new RtMidiOut();
    }
    catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return RETURN_FAILURE;
    }

    try {
        if (portNum >= midiout->getPortCount()) {
            delete midiin;
            return RETURN_FAILURE;
        }
        if (portOfMidiOut != portNum) {
            midiout->closePort();
            portOfMidiOut = portNum;
        }

        if (!midiout->isPortOpen()) {

            midiout->openPort(portNum);
           // std::cout << "midiout openPort in DLL " << portNum << std::endl;
            if (!midiout->isPortOpen()) {
               // delete midiout;
                return RETURN_FAILURE;
            }
            return RETURN_SUCCESS;
        }
    }
    catch (RtMidiError& error) {
        error.printMessage();
        delete midiout;
        return RETURN_FAILURE;
    }
}

__declspec(dllexport) void sendMidiMessage(unsigned int statusAndChannel, unsigned int
data1, unsigned int data2) {
    try {
        if (midiout == 0)
            midiout = new RtMidiOut();
    }
    catch (RtMidiError& error) {
        error.printMessage();
        return;
    }

    message[0] = statusAndChannel;
    message[1] = data1;
    message[2] = data2;
    midiout->sendMessage(&message);

    return;
}
```

Dieser Code basiert u.a. auf den Dateien qmidiin.cpp und midiout.cpp von Gary Scavone aus dem heruntergeladenen Ordner tests.

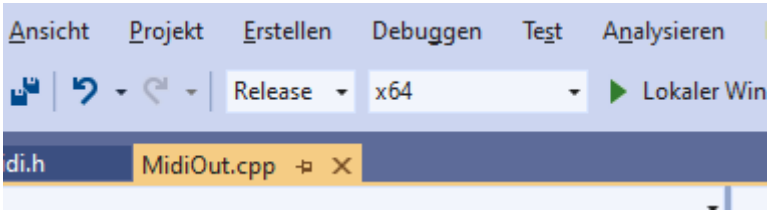
Die beiden Dateien RtMidi.h und RtMidi.cpp werden in den Projektordner des aktuellen Projekts MidifunctionsOfRtMidi kopiert und dem Projekt wie im Kapitel RtMidi unter Windows verwenden beschrieben hinzugefügt. Die Zeilen

```
#ifndef __WINDOWS_MM__
#define __WINDOWS_MM__
#endif
```

dürfen natürlich auch jetzt in RtMidi.h oben nicht fehlen. In RtMidi.cpp wird dieses Mal `#include "pch.h"` eingefügt. Außerdem wird die Zeile 2731 „error(RtMidiError::DRIVER_ERROR, errorString_);“ durch „std::cout << errorString_ << std::endl;“ ersetzt und die gleichlautende Zeile 2961 „error(RtMidiError::DRIVER_ERROR, errorString_);“ auch durch „std::cout << errorString_ << std::endl;“. Wobei sich die Positionsangaben dieser Änderungen auf die Version der Datei RtMidi.cpp von 2021 beziehen.
Der Sinn dieser Ersetzungen ist es zu verhindern, dass RtMidi sich bei Fehlern, die beim Öffnen von Ports entstehen, nicht selbst beendet, was beim aufrufenden Java-Programm zu einem Absturz führt.

Die Windows-Bibliothek winmm.lib wird im Projekt wieder als zusätzliche Abhängigkeit definiert.

Als Projektmappenkonfiguration wird „Release“ angegeben und als Projektmappen-plattform „x64“.



Achtung: Die Bibliothek winmm.lib muss für jede Konfiguration hinzufügt werden (unter Projekt → MidifunctionsOfRtMidi – Eigenschaften → Linker … s.o.).

Führe nun unter dem Menüpunkt „Erstellen“ „Projektmappe neu erstellen“ aus.

Wenn du keinen besonderen Pfad für die Projekte von Visual Studio gewählt hast, befindet sich danach im Ordner „C:\Users*Dein Name*\source\repos\MidifunctionsOfRtMidi\x64\Release“ die gewünschte DLL, die später unter Java für die programmierten Midi-Funktionalitäten verwendet wird.

Java-Beispielcode

Nun wird in der IDE Eclipse ein Beispielprojekt erstellt. Eine module-info.java Datei wird nicht angelegt.

Das Beispielprojekt besteht aus sieben Klassen. Den Code gebe ich wieder:

Klasse AppStart:

```
package de.forck.midi;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import java.awt.Dimension;
import java.awt.FlowLayout;


public class AppStart{

	private static MyMidiOut midiOutput =  new MyMidiOut();
	private static MyMidiIn midiInput = new MyMidiIn();

	public static JFrame frame = new JFrame();

	public static void main(String[] args) {
		try {
			UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		} catch (InstantiationException e) {
			e.printStackTrace();
		} catch (IllegalAccessException e) {
			e.printStackTrace();
		} catch (UnsupportedLookAndFeelException e) {
			e.printStackTrace();
		}
		showGUI();
		midiOutput.setPortOfMidiOut(0);
		setFocusToFrame();
	}

	private static void showGUI() {
		frame.setSize(new Dimension(400, 300));
		frame.setLocation(200, 200);
		frame.addKeyListener(new MyKeyListener());

		MidiOutPortChooser outDevices = new MidiOutPortChooser();
		MidiInPortChooser inDevices = new MidiInPortChooser();

		JPanel panel = new JPanel();
		panel.setLayout(new FlowLayout());
		panel.add(outDevices);
		panel.add(inDevices);

		frame.add(panel);
		frame.setVisible(true);
		frame.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
	}

	public static MyMidiOut getMidiOutput(){
		return midiOutput;
	}

	public static MyMidiIn getMidiInput() {
		return midiInput;
	}

	public static void setFocusToFrame() {
```

```
                frame.requestFocusInWindow();
        }
}
```

Klasse MidiInPortChooser

```
package de.forck.midi;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.sound.midi.MidiDevice;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.MidiUnavailableException;
import javax.swing.JComboBox;


public class MidiInPortChooser extends JComboBox<String> implements ActionListener{

        private static final long serialVersionUID = -1L;

        MyMidiIn midiIn = new MyMidiIn();
        ArrayList<String> itemsOfMidiInChooser = new ArrayList<>();

        public MidiInPortChooser(){
                initializeList();
                addActionListener(this);
        }

        public void initializeList(){
                MidiDevice device;
                MidiDevice.Info[] infos = MidiSystem.getMidiDeviceInfo();
                itemsOfMidiInChooser.add("Computer");
                for (int i = 1; i < infos.length; i++) {
                        try {
                        device = MidiSystem.getMidiDevice(infos[i]);
                        if (!itemsOfMidiInChooser.contains(infos[i].getName())
                                && !infos[i].getName().equals("Microsoft GS Wavetable Synth")
                                        && !infos[i].getName().equals("Microsoft MIDI Mapper")
                                        && !infos[i].getName().equals("Real Time Sequencer"){
                                        itemsOfMidiInChooser.add(infos[i].getName());
                                }
                        } catch (MidiUnavailableException e) {
                                e.printStackTrace();
                        }
                }
                for(String item : itemsOfMidiInChooser) {
                        this.addItem(item);
                }

        }

        public void actionPerformed(ActionEvent e){
                String devName = (String)getSelectedItem();

                for (int i = 1; i < itemsOfMidiInChooser.size(); i++) {
                        if (devName.equals(itemsOfMidiInChooser.get(i))){
                                AppStart.getMidiInput().setPort(i-1);
                                Thread thread = new Thread(AppStart.getMidiInput());
                                thread.start();
                                break;
                        }
                }
                AppStart.setFocusToFrame();
        }

}
```

Klasse MidiOutPortChooser

```
package de.forck.midi;
```

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

import javax.sound.midi.MidiDevice;
import javax.sound.midi.MidiSystem;
import javax.swing.JComboBox;


public class MidiOutPortChooser extends JComboBox<String> implements ActionListener{

	private static final long serialVersionUID = 69L;
	ArrayList<String> itemsOfMidiOutChooser = new ArrayList<>();

	public MidiOutPortChooser(){
		initializeList();
		addActionListener(this);
	}

	public void initializeList(){
		MidiDevice.Info[] infos = MidiSystem.getMidiDeviceInfo();

		for (int i = 0; i < infos.length; i++) {
			if (!itemsOfMidiOutChooser.contains(infos[i].getName())
					&& !infos[i].getName().equals("Gervill")
					&& !infos[i].getName().equals("Real Time Sequencer")
					&& !infos[i].getName().equals("Microsoft MIDI Mapper")
					){
				itemsOfMidiOutChooser.add(infos[i].getName());
			}
		}
		for(String item : itemsOfMidiOutChooser) {
			this.addItem(item);
		}
	}

	public void actionPerformed(ActionEvent e){
		String devName = (String)getSelectedItem();
		for (int i = 0; i < itemsOfMidiOutChooser.size(); i++) {
			if (devName.equals(itemsOfMidiOutChooser.get(i))){
				AppStart.getMidiOutput().setPort(i);
			}
		}
		AppStart.setFocusToFrame();
	}
}
```

Klasse MyKeyListener

```
package de.forck.midi;

import java.awt.event.*;

public class MyKeyListener implements KeyListener{

	@Override
	public void keyTyped(KeyEvent e) {
	}

	@Override
	public void keyPressed(KeyEvent e) {
		int noteNumber = 60;
		switch(e.getKeyCode()) {
			case KeyEvent.VK_C:
				noteNumber = 60;
				break;
			case KeyEvent.VK_D:
				noteNumber = 62;
				break;
```

```
                        case KeyEvent.VK_E:
                                noteNumber = 64;
                                break;
                        case KeyEvent.VK_F:
                                noteNumber = 65;
                                break;
                        case KeyEvent.VK_G:
                                noteNumber = 67;
                                break;
                        case KeyEvent.VK_A:
                                noteNumber = 69;
                                break;
                        case KeyEvent.VK_B:
                                noteNumber = 71;
                                break;
                        }
                        AppStart.getMidiOutput().sendMidiMessage(0b10010000, noteNumber, 65);
                }

                @Override
                public void keyReleased(KeyEvent e) {
                        int noteNumber = 60;
                        switch(e.getKeyCode()) {
                        case KeyEvent.VK_C:
                                noteNumber = 60;
                                break;
                        case KeyEvent.VK_D:
                                noteNumber = 62;
                                break;
                        case KeyEvent.VK_E:
                                noteNumber = 64;
                                break;
                        case KeyEvent.VK_F:
                                noteNumber = 65;
                                break;
                        case KeyEvent.VK_G:
                                noteNumber = 67;
                                break;
                        case KeyEvent.VK_A:
                                noteNumber = 69;
                                break;
                        case KeyEvent.VK_B:
                                noteNumber = 71;
                                break;
                        }
                        AppStart.getMidiOutput().sendMidiMessage(0b10000000, noteNumber, 65);
                }
}
```

Die Noten C bis B können im Programm also durch die gleichnamigen Computer-Tasten aufgerufen werden, wenn nicht ein Keyboard als Eingabegerät definiert wird.

Klasse MyMidiIn

```
package de.forck.midi;

import java.lang.invoke.MethodHandle;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.ShortMessage;

import java.lang.foreign.Linker;
import java.lang.foreign.FunctionDescriptor;
import java.lang.foreign.SymbolLookup;
import static java.lang.foreign.ValueLayout.*;


public class MyMidiIn implements Runnable{

        public boolean running = false;
        MyReceiver receiver = new MyReceiver();
```

```
	private int portNumber = 0;

	static {
		try {
			System.load("***Absoluter Pfad zur DLL***\\MidifuctionsOfRtMidi.dll");
		}catch(UnsatisfiedLinkError e) {
			System.out.println("MidifuctionsOfRtMidi.dll konnte nicht gefunden werden.");
		}
	}
	static Linker linker = Linker.nativeLinker();

	public void stop() {
		running = false;
	}

	@Override
	public void run() {
		running = true;
		ShortMessage message = new ShortMessage();
		while(running) {
			try {
				Thread.sleep(5);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
			int messageAsInt = this.receiveMidiForeignFuction(portNumber);

			try {
				message = new ShortMessage((messageAsInt >> 16)&0xF0,
							(messageAsInt >> 16)&0x0F,
							(messageAsInt >> 8) & 0xFF, messageAsInt & 0xFF);

			} catch (InvalidMidiDataException e) {
				e.printStackTrace();
			}
			if(message.getCommand() != 0) {
				receiver.send(message, 0);
			}
		}
	}

	private int receiveMidiForeignFuction(int port) {
		int value = 0;

		SymbolLookup loaderLookup = SymbolLookup.loaderLookup();
		MethodHandle receiveMidi = linker.downcallHandle(loaderLookup.find("receiveMidi").get(),
									FunctionDescriptor.of(JAVA_INT, JAVA_INT));
		try {
			value = (int)receiveMidi.invokeExact(port);
		} catch (Throwable e) {
			e.printStackTrace();
		}
		return value;
	}

	public int setPortOfMidiIn(int portNumber) {
		int value = 0;
		SymbolLookup loaderLookup = SymbolLookup.loaderLookup();
		MethodHandle setPortOfMidiIn =
				linker.downcallHandle(loaderLookup.find("setPortOfMidiIn").get(),
				FunctionDescriptor.of(JAVA_INT, JAVA_INT));
		try {
			value = (int)setPortOfMidiIn.invokeExact(portNumber);
		} catch (Throwable e) {
			e.printStackTrace();
		}
		return value;
	}
}
```

Die MyMidiIn-Klasse ist als Thread implementiert. Die run()-Methode enthält eine while-Schleife, in der alle 5 Millisekunden durch den Aufruf von receiveMidi() und anschließender

Auswertung des Rückgabewertes geschaut wird, ob ein neues Signal am aktiven Port angekommen ist. Wenn eine neue Message vorhanden ist, wird sie per receiver.send() an den beim Receiver eingestellten OutputPort gesandt.

Die Methoden receiveMidiForeignFuction(int port) und setPortOfMidiIn(int portNumber) arbeiten mit den Foreign-Function-Methoden aus dem Package java.lang.foreign. Dabei werden folgende Schritte durchlaufen:

1. Die DLL mit den aufzurufenden Methoden wird geladen. (Bereits ganz oben in der Klasse)
2. Ein Linker wird erzeugt.
3. Ein SmbolLookup wird erzeugt.
4. Ein MethodHandle wird erzeugt. Dazu ruft der oben erzeugte Linker der aktuellen Plattform die Methode downcallHandle auf. Diese Methode bekommt zwei Parameter übergeben:
    1. Die Adresse der aufzurufenden C-Methode, die mit Hilfe des SymbolLookup gefunden wird und
    2. Einen FunctionDescriptor. Die Methode of() des FunctionDescriptors erhält als ersten Parameter den Typ des Rückgabewertes und danach den Typ des Parameters der aufzurufenden Methode.
5. Mit diesem MethodHandle wird die gewünschte C-Funktion aufgerufen über die Methode invokeExact(), die die zur C-Funktion passenden Parameter übergeben bekommt.

Klasse MyMidiOut

```
package de.forck.midi;

import java.lang.invoke.MethodHandle;
import java.lang.foreign.Linker;
import java.lang.foreign.FunctionDescriptor;
import java.lang.foreign.SymbolLookup;
import static java.lang.foreign.ValueLayout.*;


public class MyMidiOut {

	static {
		try {
			System.load("***Absoluter Pfad zur DLL***\\MidifuctionsOfRtMidi.dll");
		}catch(UnsatisfiedLinkError e) {
			System.out.println("MidifuctionsOfRtMidi.dll konnte nicht gefunden werden.");
		}
	}

	static Linker linker = Linker.nativeLinker();

	public synchronized void sendMidiMessage(int statusAndChannel, int data1, int data2) {

		SymbolLookup loaderLookup = SymbolLookup.loaderLookup();
```

```
                MethodHandle sendMidiMessage =
                                linker.downcallHandle(loaderLookup.find("sendMidiMessage").get(),
                                FunctionDescriptor.ofVoid(JAVA_INT, JAVA_INT, JAVA_INT));
                try {
                        sendMidiMessage.invokeExact(statusAndChannel, data1, data2);
                } catch (Throwable e) {
                        e.printStackTrace();
                }
        }

        public int setPortOfMidiOut(int portNumber) {

                int value = 0;
                SymbolLookup loaderLookup = SymbolLookup.loaderLookup();
                MethodHandle setPortOfMidiOut =
                                linker.downcallHandle(loaderLookup.find("setPortOfMidiOut").get(),
                                FunctionDescriptor.of(JAVA_INT, JAVA_INT));
                try {
                        value = (int)setPortOfMidiOut.invokeExact(portNumber);
                } catch (Throwable e) {
                        e.printStackTrace();
                }
                return value;
        }
}
```

Klasse MyReceiver

```
package de.forck.midi;

import javax.sound.midi.MidiMessage;
import javax.sound.midi.Receiver;
import javax.sound.midi.ShortMessage;

public class MyReceiver implements Receiver{

        @Override
        public void send(MidiMessage message, long timeStamp) {
                if(message instanceof ShortMessage) {
                        AppStart.getMidiOutput().sendMidiMessage((message.getMessage()[0]),
                                                        ((ShortMessage)message).getData1(),
                                                        ((ShortMessage)message).getData2());
                }
        }

        @Override
        public void close() {}
}
```

Wenn du die Warnungen, die in Eclipse beim Start des Programms unterdrücken möchtest, musst du unter dem Menüpunkt „Run“ → „Run Configurations…“ in dem erscheinenden Dialog unter dem Reiter „(x) = Arguments“ in das Feld „VM arguments“ den Parameter `--enable-native-access=ALL-UNNAMED` eintragen.