universität
wien

BACHELORARBEIT

Titel der Bachelorarbeit

Erweiterung von NetLogo um Midi-Befehle

Verfasser:	Martin Dobiasch
Matrikel-Nummer:	0828302
Studienrichtung:	Informatikmanagement 522
Betreuer:	Ao. Univ.-Prof. Dr. Erich Neuwirth

Wien, am 18. 11. 2010

Diese Arbeit beschreibt die Arbeit rund um die Erweiterung von NetLogo [6] um Midi-Befehle.

Inhaltsverzeichnis

1 Einleitung

1.1 Auftraggeber & Betreuer

Auftraggeber und Betreuer dieser Arbeit war AO. Univ. Prof. Dr. Erich Neuwirth. Ich möchte ihm hier an dieser Stelle bei ihm für die ausgezeichnete Betreuung und Zusammenarbeit bedanken.

1.2 Ausgangslage

Bisher störte die Tatsache, dass es nicht möglich war in NetLogo Midi-Kanäle direkt anzusprechen. In der Version 4.1.1 (aktuel zum Erstellungszeitpunkt des Projektes Juli, 2010) konten lediglich die Befehle sound:play−drum drum velocity und sound:play−note instrument keynumber velocity duration verwendet werden um Modelle mit Tönen zu versehen. Das bedeutet aber große Einschränkungen. So ist es zum Beispiel nicht möglich für das Beispiel "Rettungsauto" (siehe 5.2) den Dopplereffekt zu erzeugen, geschweige denn überhaupt den Ton des Autos lauter werden zu lassen im Falle des sich nähernden Autos und wieder leiser werden zu lassen, im Fall des sich entfernden Autos.

1.3 Installation der Erweiterung

Es gibt zwei Varianten um die Erweiterung zu installieren

- die Datei "midi.jar" in einen Unterordner "midi" neben das Model legen, welches die Erweiterung verwenden möchte.
- die Datei "midi.jar" in einen Ordner "midi" in das Verzeichnis "extensions" der NetLogo-Installation legen.

Ebenso erfolgt die Installation für als Applet im Internet hinterlegte Modelle. Auch hier muss die "midi.jar" Datei ein einem "midi" Unterordner liegen.

1.4 Gliederung der Arbeit

Ich werde zuerst kurz beschreiben wie ich NetLogo um zusätzliche Befehle erweitert habe. Dann werde ich die Konzepte hinter der Erweiterung erläutern und Beispiele bringen wie die Befehle eingesetzt werden können. Diese Beispiele sind schon so vorbereitet um in einem etwaigen Unterricht mit Schülern eingesetzt werden zu können. Am Ende der Arbeit befinden sich die Listings der Erweiterung.

1.5 Test der Erweiterung

Die Arbeit wurde mittels der weiter hinten in der Arbeit beschriebenen Beispiele getestet. Getestet wurde sowohl auf Windows PCs so wie auf Apple Computern. Verwendet wurde die zum Zeitpunkt der Arbeit aktuelle NetLogo Version 4.1.1.

Es wurden bei den Tests keine Unterschiede zwischen Windows und Mac OS entdeckt.

1.6 Einsatz in der Unterrichtspraxis

Die vorliegende Erweiterung kann direkt für Unterrichtszwecke verwendet werden. Bei der Ausarbeitung der Beispiele weiter hinter hinten in der Arbeit wurde darauf geachtet sie möglichst gut aufzubereiten um sie direkt in Schulungen einsetzen zu können. Sie liegen liegen als fertig implementierte Modelle vor. Die Modelle sind nicht direkt Teil meiner Arbeit. Sie wurden von Prof. Dr. Erich Neuwirth erdacht und ich habe sie in seiner Lehrveranstaltung "Kernthemen der Fachdidaktik Informatik" im Wintersemester 2009 kennen gelernt.

1.7 Ausblick

Die Erweiterung wurde mit der Version 4.1.1 von NetLogo entwickelt. In geraumer Zeit sollte die Version 4.1.2 fertig gestellt werden. Mit dieser Version kann ein Problem meiner Erweiterung eventuell gelöst werden und das hinzufügen von Befehlen zu Notenblättern direkt als Befehl möglich sein.

Die aktuelle Implementierung könnte noch in folgenden Punkten erweitert werden:

- *Midi-Rendern* Es können zur zeit nur direkt Befehle ausgegeben werden. Eine mögliche Erweiterung ist, dass anstatt die Befehle auszugeben, die akustische Ausgabe des Modells in ein Midi-File zu schreiben.

- *Midi-Compiler* Wie später hinter in der Arbeit angemerkt wird, hat die Implementierung Befehlsabarbeitung noch Potential zur Optimierung. Es könnte angedacht werden, einzelne Blätter als Midi-Blätter zu deklarieren. Zu diesen können dann nur Midi-Befehle hinzugefügt werden, welche dann direkt als javainterne Midi-Commands kompiliert werden.

- *Protokollierung* Das Projekt könnte zusätzlich um die Funktionalität von Protokollausgaben der Befehle erweitert werden können. Anstatt von Midi-Noten werden könnten dann normale Notennamen ausgegeben werden können.

2 Erweiterung

2.1 Grundsätzliches

Eine Extentsion für NetLogo ist nichts anderes als ein Jar-Archiv. Die Erweiterung muss in einem Unterverzeichnis des Models, welches die Erweiterung verwenden will, oder in einem Unterverzeichnis des NetLogo-Programmes liegen. Jede Extension benötigt einen ClassManager. Dieser teilt NetLogo mit welche Befehle die Erweiterung mit sich bringen wird. Jeder dieser Befehle ist von org.nlogo.api.Primitive abgeleitet. Die Erweiterung muss zusätzlich ein Manifest mit den folgenden Befehlen enthalten (am Beispiel meiner Erweiterung):

```
Manifest-Version: 1.0
Extension-Name: midi
Class-Manager: at.univie.csd.MidiManager
NetLogo-Extension-API-Version: 4.1
```

2.2 Unterlagen

Für die Erstellung der Erweiterung wurden im wesentlich folgende Quellen zu Theorie zu Midi zur Rate gezogen: [1], [2] und [4] verwendet.

2.3 Struktur

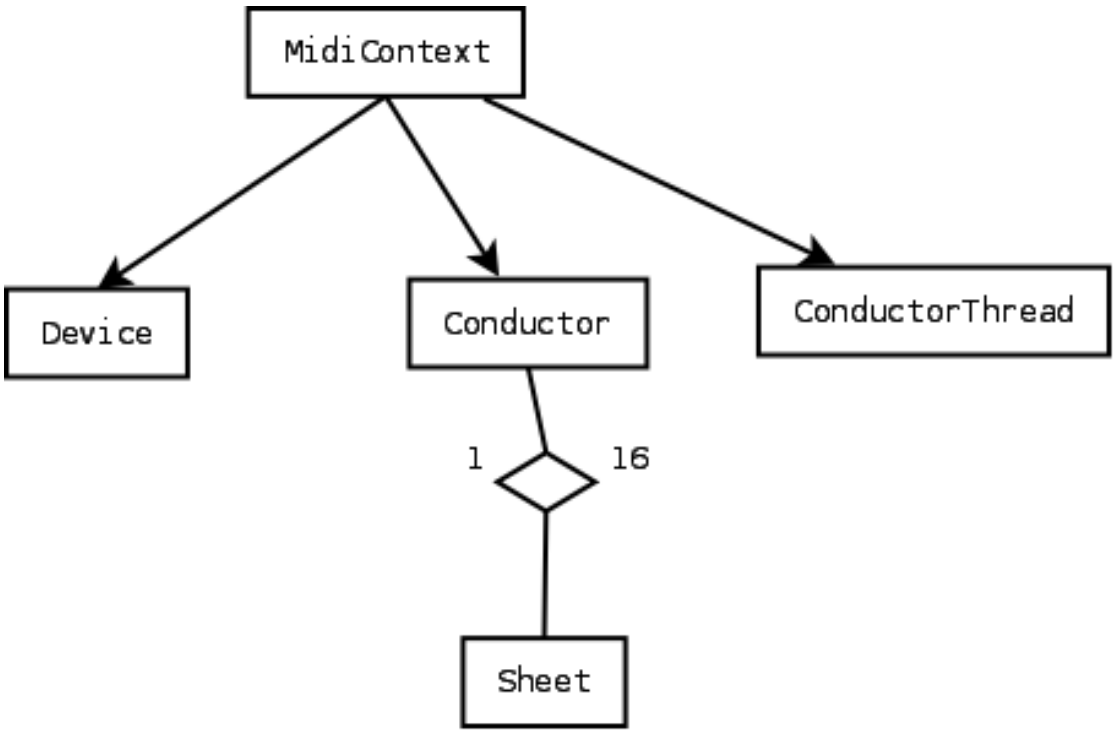

Abbildung 1: Struktur

Die Erweiterung besitzt eine relativ einfache Struktur. Beim laden der Erweiterung wird eine MidiContext Instanz erzeugt. Um eine Synchronisation der Zugriffe auf die Objekte zu ermöglichen habe ich in Anlehnung an [5] einfache Semaphore implementiert. Anstatt von Integerzählern habe ich einfach boolsche Variablen verwendet. Bei den Tests hat dies zu keinen Problemen geführt. Zu finden sind die Semaphore in der Klasse MidiContext in der Datei "MidiContext.java".

Die Implementierung als Semaphor mit Zähler würde wie folgt aussehen:

```
...
private int m_free = 1;
...
public synchronized MidiDevice getDevice()
{
  while( m_free <= 0 )
  {
    try
    {
      wait();
    }
    catch(InterruptedException e)
    {
    }
```

```
    }
    m_free--;
  }
  ...
  public synchronized void releaseDevice()
  {
    m_free++;
    notify();
  }
```

Analog die Impelentierung für den Conductor-Semaphor.

2.3.1 MidiContext

Der MidiContext soll sich um die Synchronisation der Threads kümmern. Da nur ein Device zum abspielen der MidiBefehle geöffnet wird muss der Zugriff synchronisiert werden. Um einen Befehl schreiben zu können muss zuerst der Zugriff reserviert werden. Dies geschiet über den Befehl MidiManager.getMidiContext().getDevice();. Natürlich muss auch noch überprüft werden ob das Device auch offen ist. Um das Device wieder für andere Threads wieder frei zu geben muss der Befehl MidiManager.getMidiContext().releaseDevice(); verwendet werden.

2.3.2 Conductor

Weiters beinhaltet der MidiContext eine Instanz des Conductors. Diese Klasse soll helfen einen virtuellen Dirigenten zu erstellen. Dazu hat die Klasse bis zu 16 Notenblätter. Die Anzahl ist Aufgrund der Midi-Spezifikation von 16 Kanälen festgesetzt worden. Auch hier muss wieder um die Synchronität zugewährleisten mit den Befehlen MidiManager.getMidiContext().getConductor(); und MidiManager.getMidiContext().releaseConductor(); gearbeitet werden.

2.3.3 Sheet

Die Klasse Sheet hilft bei der Verwaltung von Kommandos, die timecode gesteuert ausgeführt werden sollen. Sheet ist das Pendant zu Streams in MidiCSD [3] Jedes Notenblatt verfügt über eine Liste von Kommandos. Zu Begin der Arbeit war die Anzahl dieser Commandos auf die des MidiCSD-Toolkits beschränkt und jeder dieser Befehle wurde intern zu den entsprechenden Midi-Befehlen kompiliert. Dies wurde jedoch in Rücksprache und auf Wunsch des Betreuers so geändert, dass nun jedes beliebige Kommando hinzugefügt werden kann. Während des Programmablaufes wird also das Kommando nur gespeichert und dann NetLogo zur Ausführung übergeben. Dies führt zu kleinen Laufzeiteinbusen, erhöht jedoch aber die Flexibilität und erweitert die Möglichkeiten für den Einsatz der Erweiterung. So ist es zum Beispiel möglich den virtuellen Musikern nicht nur Noten sondern auch Bewegungen beizubringen. Der "alte" Code wurde als Kommentar im Source-Code hinterlassen.

2.3.4 ConductorThread

Damit der Dirigent im Hintergrund die Notenblätter abarbeiten kann, wurde der CondutorThread erstellt. Dieser, wenn gestartet, arbeitet Stück für Stück die Notenblätter ab. Dies macht er, in dem er sich in jedem Schritt einen Zugriff auf den Conductor reserviert um sich die anstehenden

Kommandos zubesorgen. Diese werden dann sequentiell ausgeführt. Danach legt sich der ConductorThread für 100ms schlafen um den anderen Threads nicht im Weg zu stehen.

2.4 Neue Befehle

Um einfach neue Midi-Befehle implementieren zu können habe ich eine Klasse MidiCommand geschrieben. Diese besitzt im wesentlichen zwei Methoden: preAction und postAction. Jeder neue Midi-Befehl, wenn von dieser Klasse abgeleitet, kann sich preAction Zugriff auf das Midi-Device holen und diesen dann wieder mit postAction freigeben. So kann sicher gestellt werden, dass durch neue Befehle kein Wirrwar von offenen Devices ensteht. Weiters erhöht diese Vorgangsweise die Modularität. Soll zum Beispiel etwas an der Art wie das Device geöffnet wird geändert werden, muss dies nur an einer Stelle angepasst werden.

3 Der Dirigent

Ein weiterer Punkt den ich in die Aufgabenstellung hineingenommen habe, war die Möglichkeit Sequenzen zu definieren, welche dann unabhängig abgespielbar sein sollen. Dieser Punkt wurde dann erweitert, dass es möglich sein soll, alle in NetLogo verfügbaren Befehle in so eine Sequenz zu packen.

3.1 Konzept

Wie der Name schon andeutet, ist dieser Aspekt mit der Metapher eines Orchesters gelöst worden. Ein Dirigent hat die Kontrolle über mehrere Musiker die Aktionen ausführen sollen. Er bestimmt wann das Orchester zu spielen beginnt und wann es wieder aufhört. Weiters kann er natürlich auch wieder von Vorne beginnen lassen.

3.2 Technische Realisierung

Wie schon im Konzept erwähnt wird NetLogo eine Zentrale Instanz 'Conductor' beigebracht. Diese verfügt über die folgenden Fähigkeiten:

- Alle Notenblätter löschen
- Zu einem Notenblatt etwas hinzufügen
- Zu einem Notenblatt einen mit Timecode versehenen Befehl hinzufügen
- Alles auf Anfang setzen
- Dirigieren
- Endlos spielen lassen, normal spielen lassen

3.2.1 Sheets - Kanäle

Der Dirigent hält eine fixe Anzahl von 16 Sheets. Die Zahl ist so festgelegt, da Midi 16 Kanäle besitzt. Die Midi-Kanäle besitzen die Nummern 1 - 16, die Sheets jedoch die Idizes/Nummern 0 - 15. Über eine Variable für Agents kann eine Zuordnung von Agenten zu einem Kanal gemacht werden.

```
turtles-own [channel]
...
to init
  den Turtles chanels zu ordnen
end

...

to some.procedure
  ask turtles[
    midi:instrument channel 60
  ]
end
```

Eine sehr einfach Variante den Turtles Kanäle zuzuornen ist:

```
ask turtles[
  set channel who + 1
]
```

3.2.2 Events

Die zentrale Aufgabe der Sheets ist es Befehle in chronologischer Reihenfolge auszuführen. Es gibt zwei Befehle um einem Notenblatt Befehle hinzu zufügen:

- conductor.add.to.sheet
- conductor.add.to.sheet.list

Beide Befehle machen im wesentlichen das Gleiche, jedoch werden dem Zweiten eine Liste von mit einem Timecode versehenen Befehlen übergeben, ersterer fügt einen einzelnen Befehl versehen mit einem Timecode zu einem Notenblatt hinzu. Eine genauere Beschreibung ist im Kapitel über die Befehle zu finden.

4 Befehle

Alle Befehle beginnen mit einem vorangestellten midi:. Auf eine Dokumentation der Standard-Midi-Befehle möchte ich hier verzichten. Die Dokumentation der Befehle erfolgt immer in der Form Befehl Parameter was dann in einem NetLogo-Programm als midi:Befehl Parameter geschrieben wird. Die Befehlsnamen sind nicht(!) casesensitive.

4.1 Midi-Befehle

4.1.1 Standard Befehle

- noteon channel note volume
- noteoff channel note
- instrument channel instrument (Für eine Liste an Instrumenten siehe [1]
- pitch.bend channel bend
- controller channel controller parameter
- key.pressure channel note pressure
- channel.pressure channel pressure
- volume channel volume
- expression channel expression
- modulation channel modulation
- pan channel pan
- sustain channel value
- reverb channel note volume duration
- chorus channel duration
- portamento.time channel time
- portamento channel onoff
- portamento.from channel note
- rpn channel lorpn hirpn lodata hidata
- nrpn channel lorpn hirpn lodata hidata
- reset. controllers channel
- all.notes. off channel
- pitch.sens channel semitones
- mastertune.coarse channel hidata
- mastertune.fine channel cents
- panic

4.1.2 Der Note Befehle

note channel note volume duration

*A*chtung: Dieser Befehl ist ein Blocking-Call. Er spielt eine Noten in der Dauer von duration (in ms angegeben ab). Erst dann wird das Programm forgesetzt. Er setzt zuerst die Note und legt den Thread dann schlafen, nach Abwarten des Notenwertes löscht er die Note vom entsprechenden Midi-Kanal. Dieses Verhalten kann zu Problemen führen wenn sie in Programmen in Kombination mit den Conductor-Utilities verwendet werden.

4.2 Befehle für Turtles/Agenten

updatepostion channel

Dieser Befehl setzt eine eine Position im akustischen Raum für die aktuelle Turtle, kann also nur in einem Turtle-Context angewandt werden. Die Position wird für die Turtle die den Befehl aufruft errechnet und dann als virtuelle Position auf den übergebenen Channel gelegt (siehe auch 3.2.1. Verändert werden die Parameter Pan und Expression. Ausgangspunkt für die Berechung ist der Koordinaten-Ursprung des NetLogo-Models. Als maximaler Wert für die Skalierung wird auch jeweils der maximale bzw. Wert des Koordinatensystems verwendet wird. Das heißt wenn das Model zB. auf der positiven X-Achse mehr Werte zulässt als auf der negativen. Ist die Abstufung auf der positiven Seite feiner.

4.3 Befehle für den Condcutor

4.3.1 clear.sheets

Syntax: conductor.clear.sheets

Löscht alle dem Conductor bekannten Notenblätter.

4.3.2 add.to.sheet

Syntax: conductor.add.to.sheet sheet time.distance command

Dieser Befehl fügt dem angegebenen Notenblatt am Ende ein Kommando hinzu. Das Kommando wird nach Ablauf der durch time.distance angegebenen Zeit (in ms) ausgeführt. Zusätzlich ist das der Zeitpunkt von welchem aus der Zeitpunkt für ein mögliches nachfolgendes Kommando berechnet wird. Beispiel:

```
midi:conductor.clear.sheets
midi:conductor.add.to.sheet 1 10 "midi:noteon 2 60 1"
midi:conductor.add.to.sheet 1 2000 "midi:noteoff 2 60"
midi:conductor.start
```

Nach Ablauf von zehn Milisekunden wird vom Notenblatt Eins auf dem Midi-Kanal Zwei die Note mit dem Wert 60 gespielt. Nach zwei Sekunden, also nach insgesamt 2010 Milisekunden verstummt die Note wieder.

4.3.3 add.to.sheet.list

Syntax: conductor.add.to.sheet. list sheet command.list

Dieses Kommando fügt dem angegebenen Notenblatt am Ende die Kommandos aus der Liste ein. Die Kommando-Liste hat eine einfache Syntax: [time.code commando] Die Time-Codes funktionieren analog zu denen des add.to.sheet Befehls. Das Kommando muss aber als String angegeben werden! Das kommt daher, dass NetLogo zum Entwicklungszeitpunkt (Juli-Oktober 2010) nicht ausreichend mit dem Syntax-Typ "Command" umgehen konnte. Die Befehle sollten also getestet werden bevor sie auf die Notenblätter geschrieben werden. Der Vorteil dieses Befehls ist, dass er, mit geringen Modifikationen, die Ausgabe des "to Logo" des MidiCSD-Toolkits [3] für Office Dr. Erich Neuwirth verwendet kann. Das untenstehende Beispiel demonstriert, wie der Befehl eingesetzt werden kann.

```
to trommler
  clear-all
  midi:conductor.clear.sheets

  midi:conductor.add.to.sheet.list 1 [
    [0 "midi:note 10 45 1 200"]
    [200 "midi:note 10 45 0.7 200"]
    [200 "midi:note 10 45 0.7 200"]
    [200 "midi:note 10 45 0.7 200"]
    [200 "midi:note 10 45 1 200"]
  ]
  midi:conductor.add.to.sheet.list 2 [
    [200 "midi:pan 10 -0.75"]
    [200 "midi:pan 10 -0.5"]
    [200 "midi:pan 10 -0.25"]
    [200 "midi:pan 10 0"]
  ]

  midi:conductor.add.to.sheet.list 3 [
    [5 "midi:expression 10 0.75"]
    [200 "midi:expression 10 0.69"]
    [200 "midi:expression 10 0.63"]
    [200 "midi:expression 10 0.56"]
    [200 "midi:expression 10 0.5"]
  ]

  midi:conductor.setplaymode.endless

  midi:conductor.start
end
```

4.3.4 restart

Syntax: conductor.restart

Die Position auf den Notenblättern wird wieder auf Null gesetzt. Alle Notenblätter werden also wieder von Vorne abgearbeitet.

4.3.5 conduct

Syntax: conductor.conduct

Der Dirigent wird angewiesen die aktuell anfallenden Befehle auf den Notenblättern abzuarbeiten.

4.3.6 setplaymode.endless

Syntax: conductor.setplaymode.endless

Erreicht ein "Musiker" auf seinem Notenblatt das Ende, soll er wieder von Vorne beginnen. Anders gesagt: Die Notenblätter sind nun nicht mehr linear sondern Kreise. Auf unterschiedliche Längen/Dauer der Notenblätter wird aber nicht geachtet.

4.3.7 setplaymode.normal

Syntax: conductor.setplaymode.normal

Anders als beim Endlos-Playmode wird der Dirigent angewiesen, wenn ein Notenblatt zu Ende ist, nicht wieder von Vorne zu beginnen sondern aufzuhören.

4.3.8 start

Syntax: conductor.start

Dieser Befehl weißt den Dirigenten an im Hintergrund zu dirigieren. Das Programm läuft weiter während im Hintergrund die Notenblätter abgearbeitet werden. (Siehe auch conductor.stop)

4.3.9 stop

Syntax: conductor.stop

Dieser Befehl weißt den Dirigenten an seine Arbeit zu beenden. Die Abarbeitung der Notenblätter im Hintergrund wird beendet.

5 Beispiele

Die untenstehenden Beispiele wurden gleichzeit dafür genutzt, die Implementierung der Befehle zu testen.

5.1 Rotierende Trommler

5.1.1 Idee

In diesem Beispiel sollen mehrere Trommler trommelnd im Kreis marschieren. Dazu soll unabhängig von der Musik, die sie spielen die Position verändert werden. Diese Positionsänderung soll natürlich auch hörbar sein. Es sollen also die Trommler die sich am linken Rand des Feldes aufhalten auch stärker auf der linken Seite der Lautsprecher zu hören sein sollen.

5.1.2 Benutzer-Oberfläche

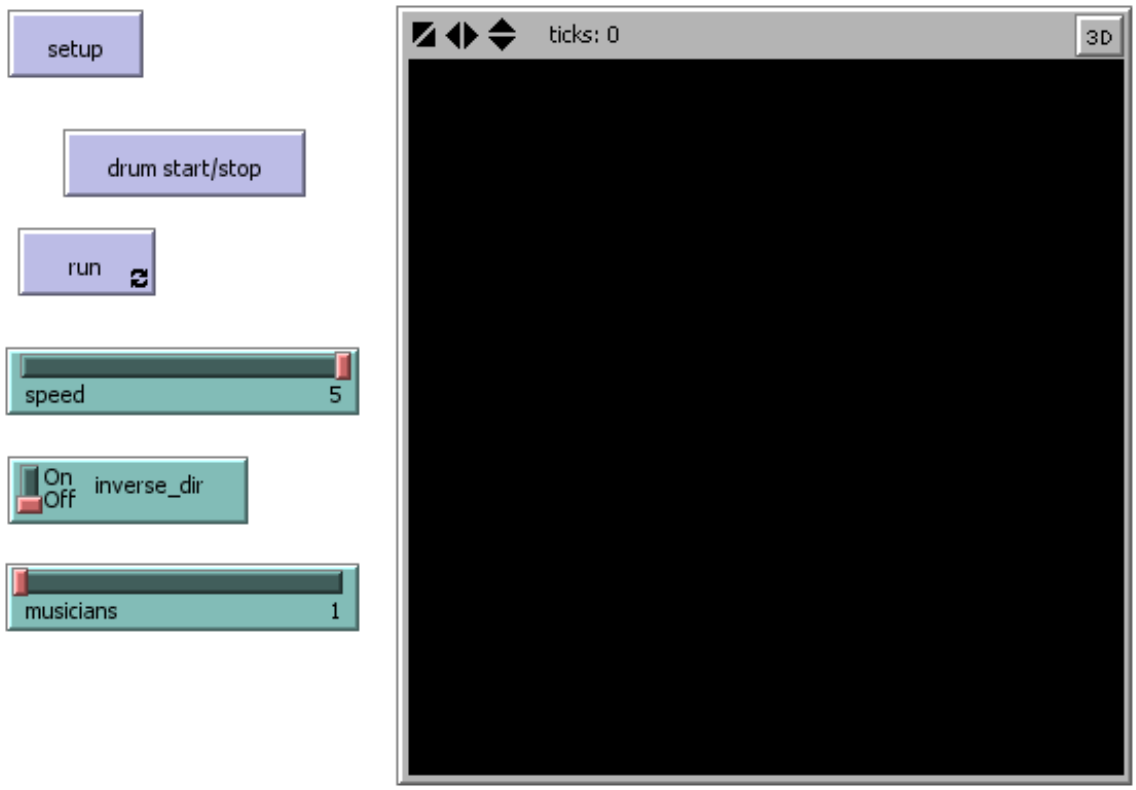

Abbildung 2: Trommler - Interface

Um das Programm zu starten ist es notwendig zuerst "Setup" und dann in beliebiger Reihenfolge "drum start/stop" und "run" zu drücken. Der Knopf "drum start/stop" bewirkt dass der/die Trommler beginnen zu trommeln , bzw. wieder aufhören zu trommeln.

Ein Druck auf "run" lässt die Trommler im Kreis marschieren, ein weiterer Druck bewirkt, dass sie stehen bleiben.

Über den Regler mit der Beschriftung speed kann die Geschwindigkeit der Trommler reguliert werden. Das kann auch passieren während die Trommler schon im Kreis marschieren.

Der Schalter "inverse_dir" Entscheidet ob die Trommler alternierend gegengleich marschieren oder nicht. Wir er auf "On" gesetzt, marschiert der erste Trommler im Uhrzeigersinn, der Zweite gegen

den Uhrzeigersinn und der Dritte wieder im Uhrzeigersinn. Wird der Schalter umgelegt während die Trommler schon marschieren verlassen diese ihre Bahnen und beginnen von ihrem aktuellen Punkt aus wieder in Kreisen zu marschieren, was an den Effekten der Akkustik aber nichts ändert.

Über den Regler "musicians" wird die Anzahl der Trommler festgelegt. Nach einer Änderung der Anzahl muss erneut auf "setup" gedrückt werden.

5.1.3 Funktionsweise

Aufgrund der Möglichkeiten, die die Erweiterung mitsich bringt ist der längste Teil des Programm, die Setup-Prozedur. Sonst besteht das Programm noch aus zwei weiteren Prozeduren: der Bewegung der Trommler und das Instruieren des Conductor.

Das Marschieren der Trommler ist simpel gehalten: Es wird lediglich berechnet, wie weit jeder einzelne Trommler vorwärts gehen muss, dann wird noch für jeden Trommler abhängig davon ob sie alternierend gegengleich marschieren berechnet wie weit er sich drehen muss. Dann wird jeder Trommler verschoben und ein midi:updateposition durchgeführt.

Das Instruieren des Conductor ist sehr einfach. Die drum-Prozedur wird, von einem Schalter am Interface aus aufgerufen. Wird noch nicht getrommelt, dh. die Variable drumming ist **false**, wird über ein midi:conductor.start der Conductor angewiesen mit dem Musikstück zu beginnen und die Variable drumming auf **true** gesetzt. Im anderen Fall, also wenn schon getrommelt wird, dh. die Variable drumming ist **true**, wird der Conductor angewiesen das Musikstück zu beenden. Das passiert über den Befehl midi:conductor.stop. Anschliesend wird noch drumming auf **false** gesetzt.

Das Setup des Models ist im Grunde auch einfach: Für drei Trommler werden die Noten und die Instrumente definiert bzw. dem Conductor mitgeteilt (Befehle midi:conductor.add.to.sheet und midi:instrument)

5.1.4 Der Source Code im Detail

Die Setup-Prozedur erstellt die Turtles die dann die Trommler darstellen und richtet das Midi-Interface sowie den Conductor ein. Es müssen die Instrumente gesetzt werden, da Kanal elf und zwölf nicht automatisch Perkussion-Instrumente verwenden. In diesem Beispiel habe ich die Variante gewählt, dass der Conductor angewiesen wird zu dirigieren, bzw. angewiesen wird mit dem Dirigieren aufzuhören. Das wird in der Prozedur drum erledigt. Der Dirigent arbeitet, nach dem er gestartet wurde im Hintergrund. (Achtung: Es sei an dieser Stelle nochmal auf die Problematik mit Blocking-Calls hingewiesen. Siehe auch 4.1.2). dosmg für Do-Something, lässt die Turtles wandern.

```
extensions [midi]

globals [drumming doit]

to setup
  clear-all

  ;; sicher ist sicher
  midi:all.notes.off 10
  midi:all.notes.off 11
  midi:all.notes.off 12
```

```
  midi:conductor.clear.sheets
  midi:conductor.setplaymode.endless

  set drumming false
  set doit false

  midi:conductor.add.to.sheet 10 10 "midi:noteon 10 45 1"
  midi:conductor.add.to.sheet 10 200 "midi:noteon 10 45 0.7"
  midi:conductor.add.to.sheet 10 200 "midi:noteon 10 45 0.7"
  midi:conductor.add.to.sheet 10 200 "midi:noteon 10 45 0.7"
  ;midi:conductor.add.to.sheet 10 200 "midi:noteoff 10 45"

  if musicians > 1 [
  midi:conductor.add.to.sheet 11 200 "midi:noteon 11 46  1"
  midi:conductor.add.to.sheet 11 200 "midi:noteon 11 46  0.7"
  midi:conductor.add.to.sheet 11 200 "midi:noteon 11 46  0.7"
  midi:conductor.add.to.sheet 11 200 "midi:noteon 11 46  0.7"
  ]

  if musicians > 2 [
  midi:conductor.add.to.sheet 12 200 "midi:noteon 12 47 1"
  midi:conductor.add.to.sheet 12 200 "midi:noteon 12 47 0.7"
  midi:conductor.add.to.sheet 12 200 "midi:noteon 12 47 0.7"
  midi:conductor.add.to.sheet 12 200 "midi:noteon 12 47 0.7"
  ]

  ;setup the instruments
  midi:instrument 11 116
  midi:instrument 12 119

  ;create the turtles
  create-turtles (9 + musicians) [
    home
    facexy max-pxcor max-pycor
    rt 45
    fd who
    ifelse inverse_dir [lt (-1 ^ who) * 90 ][lt 90]
    set size 3
    ;; thicker line is easier to see
    set pen-size 3
    ;; leave a trail
    pen-down
  ]

  ; let the unused turtles die
  ask turtle 0 [die]
  ask turtle 1 [die]
  ask turtle 2 [die]
  ask turtle 3 [die]
  ask turtle 4 [die]
  ask turtle 5 [die]
  ask turtle 6 [die]
  ask turtle 7 [die]
  ask turtle 8 [die]
end

to dosmg
  ask turtles [
    let umf who * 2 * pi
    let turn 0.4 * speed
```

```
    fd 0.001 * speed * umf

    ifelse inverse_dir
     [lt (-1 ^ who) * turn]
     [lt turn]

    ; update turtle position
    midi:updateposition (who + 1)
  ]

  tick
end

to drum
  ifelse drumming = false[
    print "starting"
    midi:conductor.start
    set drumming true
  ]
  [
    print "stopping"
    midi:conductor.stop
    set drumming false
    set doit false
  ]
end
```

5.1.5 Ausblick

Ziel des Models ist es die Funktion "midi:updateposition" und den Conductor auf seine "start/stop"-Fähigkeit zu testen. Aus diesem Grund ist das Model sehr einfach gehalten. Ein paar Varianten um das Model zu erweitern sind:

- *Orchester* Im aktuellen Model marschieren nur ein bis drei Trommler im Kreis. Eine Erweiterung ist weitere Musiker mitspielen zu lassen. Das kann über verschiedene Wege passieren:
  - *komplett-statisch* Im Programm-Code wird festgelegt was ein einzelner Musiker, wenn ausgewählt, spielt. Das Interface wird um Switches für die einzelnen Musiker erweitert.
  - *teil-statisch* Das Interface wird erweitert, um die Möglichkeit für eine festgelegt Anzahl an Musikern, Noten und deren Werte einzugeben. Der Programm-Code muss diese Werte dann in die Notenblätter schreiben und dem Conductor mitteilen. Achtung: Die Erweiterung des Interfaces um die Eingabe ist bei weitem nicht einfach, da eine vernünftige Struktur gefunden werden muss, wie die Noten und ihre Werte eingeben werden können. Auch im Programm-Code muss dann einiges an Abfragen passieren ob einzelne Felder gesetzt sind
  - *dynamisch* Die Musiker und ihre Noten werden über Dateien definiert. Das Programm wird so erweitert, dass es in einem Ordner nach Dateien mit einem normierten Dateinamen sucht und aus diesen die Noten und ihre Noten-Werte ausliest. Als Format für die Dateien ist ein einfaches CSV-Format ausreichend.
- *Marsch-Formationen* Anstatt die Trommler nur in Kreisbahnen gehen zu lassen, kann das Model um die Funktionalität erweitert werden, dass sich die Trommler auf selbst definierten Funktions-Bahnen bewegen. So kann das Interface um Eingabefelder erweitert werden wo für

jeden Trommler die Grenzen für die Funktion und die Funktion, welche abhänig von einem Zeitparameter sein sollte, eingegeben werden kann. Als programmtechnische Lösung kann zB. in der Setuproutine eine Liste mit Positionsänderungen angelegt werden, diese wird durch schrittweise Berechnung, der einzelnen vom Benutzer eingegebenen Funktionen berechnet.

5.2 Rettungsauto

5.2.1 Idee

Ein simuliertes Auto soll über den Bildschirm fahren und dabei das Martinshorn erklingen lassen. Es soll aber ein realistisches Modell sein, also der Dopplereffekt simuliert werden.

5.2.2 Benutzer-Oberfläche

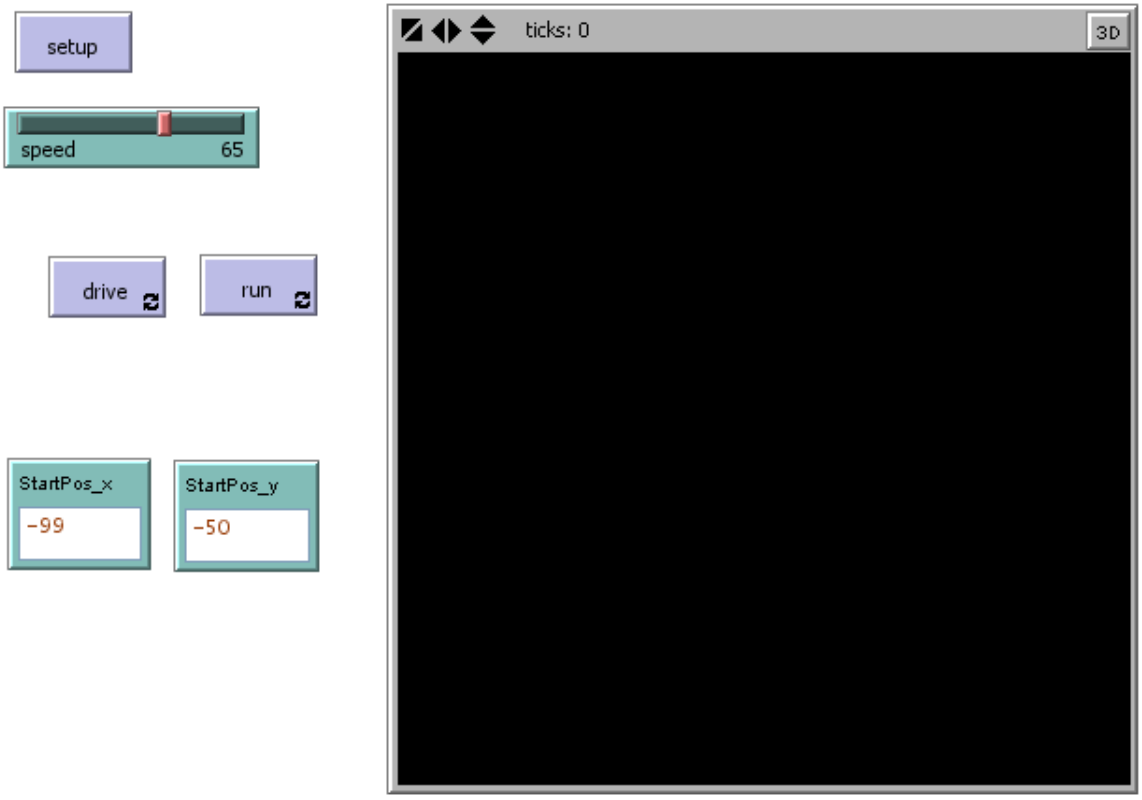

Abbildung 3: Rettungsauto - Interface

Die Abbildung 3 zeigt die Oberfläche für die Anwender des Modells. Links oben befindet sich der Knopf um das Modell zu initialisieren. Um die das Rettungsautofahren zulassen müssen die Knöpfe "run" und "drive" gedrückt werden. Drive lässt das Auto über die Fläche fahren, während "run" das Martinshorn aktiviert. Weiter unten befindet sich ein Regler um die Geschwindigkeit des Autos zu regeln. Sowie zwei Felder mit welchen die Startkoordinaten des Autos festgelegt werden können. Durch ändern dieser Koordinaten ändert sich auch die Richtung in welche sich das Auto bewegt. Vorausgesetzt es wird das Modell nach einer Änderung der Koordinaten neu initialisiert.

5.2.3 Funktionsweise

Das Grundprinzip des Models ist einfach: Über die zwei Knöpfe werden unabhängig von einander zwei Prozeduren wiederholt ausgeführt. Die Prozedur für das Martinshorn enthält lediglich die Anweisung conductor.conduct um das Abspielen der Midi-Befehle für das Martinshorn zu bewirken. Die zweite Procedur berechnet die Positionsänderung anhand der Parameter Geschwindigkeit und vergangener Zeit. Anschließen schiebt sie die Schildkröte, welche das Rettungsauto repräsentiert nach vorne und führt ein Positions-Update für die Schildkröte durch.

5.2.4 Der Source Code im Detail

Die Setup-Prozedur erstellt das virtuelle Rettungsauto und richtet den Conductor ein. Als Hilfsfunktion verwendet sie make.tatue um die Töne für das Rettungsauto zu erzeugen. Das Beispiel verwendet in diesem Fall, als Unterschied zum Trommler-Beispiel die Variante den Conductor in jedem Schritt einzeln aufzurufen und ihn dirigen lassen. Damit kann das Auto unabhängig von seinem Martinshorn fahren. Das Dirigieren wird in der kurzen Prozedur runit realisiert, das Fahren in drive.

```
extensions [midi]

globals [lasttick velocity lasttime]

turtles-own[channel]

to setup
  clear-all

  midi:conductor.clear.sheets
  midi:all.notes.off 1

  create-turtles 1[
    set size 10
    setxy StartPos_x StartPos_y
    facexy 0 0
    set channel who + 1
    midi:updateposition channel
  ]

  make.tatue

  reset-timer
  set lasttime 0
end

to make.tatue
  midi:conductor.add.to.sheet 1 10 "midi:noteon 1 60 1"
  midi:conductor.add.to.sheet 1 250 "midi:noteoff 1 60"
  midi:conductor.add.to.sheet 1 10 "midi:noteon 1 65 1"
  midi:conductor.add.to.sheet 1 250 "midi:noteoff 1 65"

  midi:instrument 1 57

  midi:conductor.setplaymode.endless
end
```

```
to runit
  midi:conductor.conduct
end

to drive
  ask turtle 0 [
    ifelse timer - lasttime < 2 [
      fd speed * 0.4 * (timer - lasttime) * (ticks - lasttick) ]
    [
      fd speed * 0.4 ]

    midi:updateposition channel

    if xcor < 0 [
      midi:pitch.bend channel ( 0.7 / min-pxcor * xcor * -1 )
    ]
    if xcor = 0 [ midi:pitch.bend channel 0]
    if xcor > 0 [
      midi:pitch.bend channel ( 0.7 / max-pxcor * xcor * -1 )
    ]
  ]

  set lasttick ticks
  set lasttime timer
  tick
end
```

5.2.5 Ausblick

Das Modell zeigt zur Zeit nur elementare Fähigkeiten der Midi-Extension, kann aber natürlich beliebig erweitert werden.

- *Mehr Fahrzeuge:* Das Modell kann natürlich um weitere Autos erweitert werden. So ist es möglich über den Dirigenten Notenblätter für die verschiedenen Fahrzeuge anzulegen. In der Fahrprozedur müsste dann lediglich der Code erweitert werden, so das mehrere Agenten bewegt werden und ihre Sound-Position aktualisieren. Weiters muss natürlich auch für die weiteren Fahrzeuge der Dopplereffekt generiert werden.

- *Fahrtverläufe:* In der aktuellen Implementierung kann lediglich der Startpunkt des Autos gesetzt werden, nicht jedoch der Verlauf der Fahrt. Ein mögliche erweiterung ist, dass der Benutzer aus mehreren Funktionen auswählen oder diese auch selber eingeben kann, welche den Verlauf der Fahrt berechnen.

- *Genauerer Dopplereffekt:* Der Dopplereffekt wird im vorliegenden Modell nur approximiert. Durch Literatur-Recherche und geringfügige Änderungen am Code, kann sehr einfach eine bessere Annäherung erreicht werden.

6 Source Code

6.1 Cmd.java

```
package at.univie.csd;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;

public class Cmd extends DefaultCommand
{

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    System.out.println("Cmd executed");
  }

}
```

6.2 AllNotesOff.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiCommand;

public class AllNotesOff extends MidiCommand
{
  public AllNotesOff()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
```

```
    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 123,0);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.3 ChannelPressure.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class ChannelPressure extends MidiCommand
{
  public ChannelPressure()
  {
    super();
  }

  public Syntax getSyntax()
  {
```

```
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER,
        Syntax.TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double rval;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      rval= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CHANNEL_PRESSURE, channel - 1, (int)
          Conversion.Rescale(rval, 0, 1, 0, 127), 0);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.4 Chorus.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;
```

```
import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class Chorus extends MidiCommand
{
  public Chorus()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double val;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      val= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 93, (int)
          Conversion.Rescale(val, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.5 ConductorAddToSheet.java

```
/**
 *
 */
package at.univie.csd.command;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiManager;
import at.univie.csd.midi.Conductor;
import at.univie.csd.midi.Sheet;


/**
 * @author Martin Dobiasch
 *
 */
public class ConductorAddToSheet extends DefaultCommand
{
  public Syntax getSyntax()
  {
    return Syntax.commandSyntax( new int[] {Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER,
        Syntax.TYPE_STRING} ) ;
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
  LogoException
  {
    Sheet[] sheets;
    int sheet;
    String cmd;
    //double[] cargs;
    int dur;
    // int i;
    // LogoList list;

    try
    {
      sheet= args[0].getIntValue() - 1;
      cmd= args[2].getString();
      dur= args[1].getIntValue();
      /*list= args[2].getList();

      cargs= new double[list.size() + 1];
      //cargs[0]= sheet;
      i= 0;
      while( ! list.isEmpty())
      {
        cargs[i]= (Double) list.first();
        list= list.butFirst();
        i++;
      }*/
    }
```

```
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    Conductor c= MidiManager.getMidiContext().getConductor();
    sheets= c.getSheets();

    if( sheets[sheet] == null )
      sheets[sheet]= new Sheet();

    //sheets[sheet].addCommand(cmd, cargs, Sheet.NO_TIMECODE);
    sheets[sheet].addCommand(cmd, dur);

    MidiManager.getMidiContext().releaseConductor();

    // throw new ExtensionException( cmd );
  }
}
```

6.6 ConductorAddToSheetWT.java

```
package at.univie.csd.command;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.LogoList;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiManager;
import at.univie.csd.midi.Conductor;
import at.univie.csd.midi.Sheet;

public class ConductorAddToSheetWT extends DefaultCommand
{
  public Syntax getSyntax()
  {
    return Syntax.commandSyntax( new int[] {Syntax.TYPE_NUMBER, Syntax.
        TYPE_LIST } ) ;
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
  LogoException
  {
    Sheet[] sheets;
    int sheet;
    LogoList list;
    LogoList cmd; //command
    String cmds; //command string
    String cmdn = null;
    long tc;
```

```
    try
    {
      sheet= args[0].getIntValue() - 1;
      list= args[1].getList();

      Conductor c= MidiManager.getMidiContext().getConductor();
      sheets= c.getSheets();

      if( sheets[sheet] == null )
        sheets[sheet]= new Sheet();

      while( ! list.isEmpty())
      {
        cmd= (LogoList) list.first();//get next Command

        tc= ((Double) cmd.first()).longValue(); //get Timecode
        cmd= cmd.butFirst(); //go to command to add

        //cmds= (String) ((LogoList)cmd.first()).first(); //get command to
            add
        cmds= (String) cmd.first(); //get command to add

        sheets[sheet].addCommand(cmds, tc);

        list= list.butFirst();
      }


    }
    catch(LogoException e)
    {
      throw new ExtensionException(cmdn + e.getMessage());
    }

    MidiManager.getMidiContext().releaseConductor();
  }
}
```

6.7 ConductorClearSheets.java

```
package at.univie.csd.command;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiManager;
import at.univie.csd.midi.Conductor;

public class ConductorClearSheets extends DefaultCommand
{
  public Syntax getSyntax()
  {
```

```
    return Syntax.commandSyntax( new int[] { } ) ;
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
  LogoException
  {
    Conductor c= MidiManager.getMidiContext().getConductor();

    c.clearSheets();

    c.resettime();

    MidiManager.getMidiContext().releaseConductor();
  }
}
```

6.8 ConductorConduct.java

```
package at.univie.csd.command;

import java.util.ArrayList;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiManager;
import at.univie.csd.midi.Conductor;
import at.univie.csd.midi.TimedEvent;

public class ConductorConduct extends DefaultCommand/*MidiCommand*/
{
  private class WorkerThread extends Thread
  {
    private Context ctx;
    private ArrayList<TimedEvent> events;

    public WorkerThread(Context ctx, ArrayList<TimedEvent> events)
    {
      this.ctx= ctx;
      this.events= events;
    }

    public void run()
    {
      TimedEvent ev;
      for(int i= 0; i < events.size(); i++)
      {
        ev= events.get(i);
        //throw new ExtensionException( ev.msg );

        try
```

```
        {
          //Runs a Command, and not waits for the command to terminate
          ctx.runCommand(ev.msg, false);
        }
        catch (ExtensionException e)
        {
          // TODO Auto-generated catch block
          e.printStackTrace();
        }
      }
    }
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax( new int[] {} ) ;
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
  LogoException
  {
    Conductor c= MidiManager.getMidiContext().getConductor();
    // TimedEvent ev;
    ArrayList<TimedEvent> l;
    // int i= 0;

    l= c.getDueEvents();

    MidiManager.getMidiContext().releaseConductor();

    new WorkerThread(ctx,l).run();
    // As we are not 'doing' commands we don't need to get the
    //   MidiContext here
    // preAction();
    // MidiManager.debug("Conductor conduct start");
    /*
    for(i= 0; i < l.size(); i++)
    {
      ev= l.get(i);
      //throw new ExtensionException( ev.msg );

      //Runs a Command, and not waits for the command to terminate
      ctx.runCommand(ev.msg, false);
      // ctx.runCommand(ev.msg, true);

      //m_recv.send(ev.msg, ev.abstime);
    }
    */
    // postAction();
    // MidiManager.debug("Conductor conduct end");
    // MidiManager.getMidiContext().releaseConductor();
    // MidiManager.debug("Conductor conduct MidiContext released");
  }
}
```

6.9 ConductorPlaymodeEndless.java

```
/**
 *
 */
package at.univie.csd.command;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiManager;
import at.univie.csd.midi.Conductor;

/**
 * @author Martin Dobiasch
 *
 */
public class ConductorPlaymodeEndless extends DefaultCommand
{
  public Syntax getSyntax()
  {
    return Syntax.commandSyntax( new int[] { } ) ;
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
  LogoException
  {
    Conductor c= MidiManager.getMidiContext().getConductor();

    c.setPlayMode( Conductor.ENDLESS );

    MidiManager.getMidiContext().releaseConductor();
  }
}
```

6.10 ConductorPlaymodeNormal.java

```
/**
 *
 */
package at.univie.csd.command;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiManager;
import at.univie.csd.midi.Conductor;
```

```
/**
 * @author Martin Dobiasch
 *
 */
public class ConductorPlaymodeNormal extends DefaultCommand
{
  public Syntax getSyntax()
  {
    return Syntax.commandSyntax( new int[] { } ) ;
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
  LogoException
  {
    Conductor c= MidiManager.getMidiContext().getConductor();

    c.setPlayMode( Conductor.NORMAL );

    MidiManager.getMidiContext().releaseConductor();
  }
}
```

6.11 ConductorReset.java

```
package at.univie.csd.command;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiManager;
import at.univie.csd.midi.Conductor;

public class ConductorReset extends DefaultCommand
{
  public Syntax getSyntax()
  {
    return Syntax.commandSyntax( new int[] {} ) ;
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
  LogoException
  {
    Conductor c= MidiManager.getMidiContext().getConductor();

    c.resetSheets();

    c.resettime();

    MidiManager.getMidiContext().releaseConductor();
```

```
  }
}
```

6.12 ConductorStart.java

```
package at.univie.csd.command;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiContext;
import at.univie.csd.MidiManager;

public class ConductorStart extends DefaultCommand/*MidiCommand*/
{
  public Syntax getSyntax()
  {
    return Syntax.commandSyntax( new int[] {} ) ;
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
  LogoException
  {
    MidiContext mctx;

    // MidiManager.debug("conductor.start start");

    mctx= MidiManager.getMidiContext();

    mctx.getConductor().start();
    mctx.releaseConductor();

    mctx.startConductor(ctx);

    // MidiManager.debug("conductor.start end");
  }
}
```

6.13 ConductorStop.java

```
package at.univie.csd.command;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
```

```
import org.nlogo.api.Syntax;

import at.univie.csd.MidiManager;

public class ConductorStop extends DefaultCommand/*MidiCommand*/
{
  public Syntax getSyntax()
  {
    return Syntax.commandSyntax( new int[] {} ) ;
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
  LogoException
  {
    // MidiManager.debug("conductor.stop start");
    MidiManager.getMidiContext().stopCondcutor();
    // MidiManager.debug("conductor.stop end");
  }
}
```

6.14 Controller.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiCommand;

public class Controller extends MidiCommand
{
  public Controller()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER,
         Syntax.TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
       LogoException
  {
    int channel;
    int contr;
```

```
    int ival;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      contr= args[1].getIntValue();
      ival= (int) args[2].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, contr, ival);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.15 Expression.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class Expression extends MidiCommand
{
  public Expression()
  {
    super();
  }
```

```
  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double vol;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      vol= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 11, (int)
          Conversion.Rescale(vol, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.16 Instrument.java

```
/**
 *
 */
package at.univie.csd.command;

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

```
import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiCommand;

/**
 * @author Martin
 *
 */
public class Instrument extends MidiCommand
{
  public Instrument()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER,
        Syntax.TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    int inst;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      inst= args[1].getIntValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.PROGRAM_CHANGE, channel - 1, inst - 1, 0);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
```

```
  }
}
```

6.17 KeyPressure.java

```
/**
 *
 */
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

/**
 * @author Martin
 *
 */
public class KeyPressure extends MidiCommand
{
  public KeyPressure()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER,
        Syntax.TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    int note;
    double rval;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      note= args[1].getIntValue();
      rval= args[2].getDoubleValue();
    }
```

```
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CHANNEL_PRESSURE, channel - 1, note, (int)
          Conversion.Rescale(rval, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.18 MastertuneCoarse.java

```
/**
 *
 */
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

/**
 * @author Martin
 *
 */
public class MastertuneCoarse extends MidiCommand
{
  public MastertuneCoarse()
  {
    super();
  }

  public Syntax getSyntax()
```

```
{
  return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
      TYPE_NUMBER});
}

public void perform(Argument[] args, Context ctx) throws
    ExtensionException,
    LogoException
{
  int channel;
  int lorpn;
  int hirpn;
  int lodata;
  int hidata;

  preAction();
  ShortMessage msg= new ShortMessage();
  try
  {
    channel= args[0].getIntValue();
    lorpn= 2;
    hirpn= 0;
    lodata= 0;
    hidata= args[1].getIntValue() + 64;
  }
  catch(LogoException e)
  {
    throw new ExtensionException(e.getMessage());
  }

  try
  {
    msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 101, (int)
        Conversion.Rescale(hirpn, 0, 1, 0, 127));
    m_recv.send(msg, -1);
    msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 100, (int)
        Conversion.Rescale(lorpn, 0, 1, 0, 127));
    m_recv.send(msg, -1);
    msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1,   6, (int)
        Conversion.Rescale(hidata, 0, 1, 0, 127));
    m_recv.send(msg, -1);
    msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1,  38, (int)
        Conversion.Rescale(lodata, 0, 1, 0, 127));
    m_recv.send(msg, -1);
    msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 101, 127);
    m_recv.send(msg, -1);
    msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 100, 127);
    m_recv.send(msg, -1);
  }
  catch(InvalidMidiDataException e)
  {
    throw new ExtensionException("Invalid␣Midi␣data␣" + e.getMessage());
  }
  finally
  {
    postAction();
  }

  postAction();
}
```

```
}
```

6.19 MastertuneFine.java

```
/**
 *
 */
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;
import at.univie.csd.TwoByte;

/**
 * @author Martin
 *
 */
public class MastertuneFine extends MidiCommand
{
  public MastertuneFine()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER});
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    int lorpn;
    int hirpn;
    int lodata;
    int hidata;
    int cents;
    TwoByte tb;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
```

```
      cents= args[1].getIntValue();
      tb= Conversion.TwoByteMidiRescale(cents, -100, 100);

      lorpn= 1;
      hirpn= 0;
      lodata= tb.lsb;
      hidata= tb.msb;
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 101, (int)
          Conversion.Rescale(hirpn, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 100, (int)
          Conversion.Rescale(lorpn, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1,   6, (int)
          Conversion.Rescale(hidata, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1,  38, (int)
          Conversion.Rescale(lodata, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 101, 127);
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 100, 127);
      m_recv.send(msg, -1);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    postAction();
  }
}
```

6.20 Modulation.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
```

```
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class Modulation extends MidiCommand
{
  public Modulation()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double val;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      val= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 1, (int)
          Conversion.Rescale(val, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.21 Note.java

```
/**
 *
 */
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

/**
 * @author Martin Dobiasch
 *
 */
public class Note extends MidiCommand
{

  public Note()
  {
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER,
        Syntax.TYPE_NUMBER, Syntax.TYPE_NUMBER, Syntax.TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    int note;
    double vol;
    int dur;

    preAction();
    ShortMessage msg= new ShortMessage();
    // msg.setMessage(ShortMessage.NOTE_ON,
    try
    {
      channel= args[0].getIntValue();
      note= args[1].getIntValue();
      vol= args[2].getDoubleValue();
      dur= args[3].getIntValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
```

```
    {
      msg.setMessage(ShortMessage.NOTE_ON, channel - 1, note, (int)
          Conversion
          .Rescale(vol, 0, 1, 0, 127));
      m_recv.send(msg, -1);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    postAction();
    try
    {
      Thread.sleep(dur);
      preAction();
      msg.setMessage(ShortMessage.NOTE_OFF, channel - 1, note, (int)
          Conversion
          .Rescale(vol, 0, 1, 0, 127));
      m_recv.send(msg, -1);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    catch(InterruptedException e)
    {
      throw new ExtensionException("Interruped " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    postAction();
  }
}
```

6.22 NoteOff.java

```
/**
 *
 */
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
```

```
import org.nlogo.api.Syntax;

import at.univie.csd.MidiCommand;

/**
 * @author Martin Dobiasch
 *
 */
public class NoteOff extends MidiCommand
{

  public NoteOff()
  {
    //super(ctx);
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER,
        Syntax.TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    int note;

    preAction();
    ShortMessage msg= new ShortMessage();
    // msg.setMessage(ShortMessage.NOTE_ON,
    try
    {
      channel= args[0].getIntValue();
      note= args[1].getIntValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.NOTE_OFF, channel - 1, note, 0);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.23 NoteOn.java

```
/**
 *
 */
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

/**
 * @author Martin Dobiasch
 *
 */
public class NoteOn extends MidiCommand
{

  public NoteOn()
  {
    //super(ctx);
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER,
        Syntax.TYPE_NUMBER, Syntax.TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    int note;
    double vol;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      note= args[1].getIntValue();
      vol= args[2].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }
```

```
    try
    {
      msg.setMessage(ShortMessage.NOTE_ON, channel - 1, note, (int)
          Conversion
          .Rescale(vol, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.24 Nrpn.java

```
/**
 *
 */
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

/**
 * @author Martin
 *
 */
public class Nrpn extends MidiCommand
{
  public Nrpn()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER,
```

```
        Syntax.TYPE_NUMBER, Syntax.TYPE_NUMBER, Syntax.TYPE_NUMBER});
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    int lorpn;
    int hirpn;
    int lodata;
    int hidata;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      lorpn= args[1].getIntValue();
      hirpn= args[2].getIntValue();
      lodata= args[3].getIntValue();
      hidata= args[4].getIntValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 99, (int)
          Conversion.Rescale(hirpn, 0, 1, 0, 127));
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 98, (int)
          Conversion.Rescale(lorpn, 0, 1, 0, 127));
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1,  6, (int)
          Conversion.Rescale(hidata, 0, 1, 0, 127));
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 38, (int)
          Conversion.Rescale(lodata, 0, 1, 0, 127));
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 99, 127);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 98, 127);

    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.25 Pan.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class Pan extends MidiCommand
{
  public Pan()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double vol;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      vol= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 10, (int)
          Conversion.Rescale(vol, -1, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }
```

```
    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.26 Panic.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiCommand;

public class Panic extends MidiCommand
{
  public Panic()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;

    preAction();
    ShortMessage msg= new ShortMessage();

    try
    {
      for(channel= 0; channel < 16; channel++)
      {
        msg.setMessage(ShortMessage.CONTROL_CHANGE, channel, 121, 0);
        m_recv.send(msg, -1);
        msg.setMessage(ShortMessage.CONTROL_CHANGE, channel, 123,0);
        m_recv.send(msg, -1);
      }
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
```

```
    }
    finally
    {
      postAction();
    }

    postAction();
  }
}
```

6.27 PitchBend.java

```
/**
 *
 */
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;
import at.univie.csd.TwoByte;

/**
 * @author Martin Dobiasch
 *
 */
public class PitchBend extends MidiCommand
{
  public PitchBend()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER,
        Syntax.TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double rval;
    TwoByte MyTwobyte;;

    preAction();
```

```
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      rval= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    MyTwobyte = Conversion.TwoByteMidiRescale(rval, -1, 1);

    try
    {
      msg.setMessage(ShortMessage.PITCH_BEND, channel - 1, MyTwobyte.lsb,
          MyTwobyte.msb);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.28 PitchSens.java

```
/**
 *
 */
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

/**
 * @author Martin Dobiasch
 *
 */
```

```
public class PitchSens extends MidiCommand
{
  public PitchSens()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    int semitones;
    ShortMessage msg= new ShortMessage();

    preAction();
    try
    {
      channel= args[0].getIntValue();
      semitones= args[1].getIntValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 101, (int)
          Conversion.Rescale(0, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 100, (int)
          Conversion.Rescale(0, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1,   6, (int)
          Conversion.Rescale(semitones, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1,  38, (int)
          Conversion.Rescale(0, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 101, 127);
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 100, 127);
      m_recv.send(msg, -1);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }
```

```
    postAction();
  }
}
```

6.29 Portamento.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class Portamento extends MidiCommand
{
  public Portamento()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double onoff;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      onoff= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 65, (int)
          Conversion.Rescale(onoff, 0, 1, 0, 127));
```

```
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.30 PortamentoFrom.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiCommand;

public class PortamentoFrom extends MidiCommand
{
  public PortamentoFrom()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    int note;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
```

```
      note= args[1].getIntValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 84, note);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.31 PortamentoTime.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class PortamentoTime extends MidiCommand
{
  public PortamentoTime()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }
```

```
  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double time;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      time= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 5, (int)
          Conversion.Rescale(time, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.32 ResetControllers.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.MidiCommand;

public class ResetControllers extends MidiCommand
{
```

```
  public ResetControllers()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 121, 0);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.33 Reverb.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
```

```
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class Reverb extends MidiCommand
{
  public Reverb()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double val;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      val= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 91, (int)
          Conversion.Rescale(val, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.34 Rpn.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class Rpn extends MidiCommand
{
  public Rpn()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER,
         Syntax.TYPE_NUMBER, Syntax.TYPE_NUMBER, Syntax.TYPE_NUMBER});
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
       LogoException
  {
    int channel;
    int lorpn;
    int hirpn;
    int lodata;
    int hidata;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      lorpn= args[1].getIntValue();
      hirpn= args[2].getIntValue();
      lodata= args[3].getIntValue();
      hidata= args[4].getIntValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
```

```
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 101, (int)
          Conversion.Rescale(hirpn, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 100, (int)
          Conversion.Rescale(lorpn, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1,   6, (int)
          Conversion.Rescale(hidata, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1,  38, (int)
          Conversion.Rescale(lodata, 0, 1, 0, 127));
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 101, 127);
      m_recv.send(msg, -1);
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 100, 127);
      m_recv.send(msg, -1);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    postAction();
  }
}
```

6.35 Sustain.java

```
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class Sustain extends MidiCommand
{
  public Sustain()
  {
    super();
  }

  public Syntax getSyntax()
  {
```

```
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double val;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      val= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 64, (int)
          Conversion.Rescale(val, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.36 UpdatePosition.java

```
package at.univie.csd.command;

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

import org.nlogo.agent.Turtle;
import org.nlogo.agent.World;
import org.nlogo.api.Agent;
import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
```

```
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

public class UpdatePosition extends MidiCommand
{
  public UpdatePosition()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[]{Syntax.TYPE_NUMBER});
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double tpan;
    int pan;
    //double tvol;
    //int vol;
    double texp;
    int exp;
    Agent agent;
    World world;
    Turtle t;
    double px, py;

    // MidiManager.debug("UpdatePosition start");

    agent= ctx.getAgent();

    channel= (int) agent.id();

    world= (World) agent.world();
    t= world.getTurtle(channel);

    //Debugging
    /*MidiManager.debug("Update Position: \n" +
    "x: " + t.xcor() + "\n" +
    "y: " + t.ycor() + "\n" +
    "xmin: " + world.minPxcor() + "\n" +
    "xmax: " + world.maxPxcor() + "\n" +
    "ymin: " + world.minPycor() + "\n" +
    "ymax: " + world.maxPycor() );*/

    px= t.xcor();
    py= t.ycor();

    channel= args[0].getIntValue();

    if( px > 0 )
      tpan= world.maxPxcor() / px;
    else if( px < 0 )
```

```
      tpan= ( world.minPxcor() / px ) * -1;
    else
      tpan= 0;

    /*if( py > 0 )
      tvol= world.maxPycor() / py;
    else if( py < 0 )
      tvol= ( world.minPycor() / py );
    else
      tvol= 1;*/

    if( py > 0 )
      texp= ( world.maxPycor() / py ) / 2 + 0.5;
    else if( py < 0 )
      texp= 0.5 - ( ( world.minPycor() / py ) / 2 );
    else
      texp= 0.5;

    pan= (int) Conversion.Rescale(tpan, -1, 1, 0, 127);
    //vol= (int) Conversion.Rescale(tvol, 0, 1, 0, 127);
    exp= (int) Conversion.Rescale(texp, 0, 1, 0, 127);

    ShortMessage msg1= new ShortMessage();
    ShortMessage msg2= new ShortMessage();

    try
    {
      msg1.setMessage(ShortMessage.CONTROL_CHANGE, channel, 10, pan);
      //msg2.setMessage(ShortMessage.CONTROL_CHANGE, channel, 7, vol);
      msg2.setMessage(ShortMessage.CONTROL_CHANGE, channel, 11, exp);

      // MidiManager.debug("UpdatePosition pre preAction");
      preAction();
      // MidiManager.debug("UpdatePosition post preAction");

      m_recv.send(msg1, -1);
      m_recv.send(msg2, -1);

      // MidiManager.debug("UpdatePosition pre postAction");
      postAction();
      // MidiManager.debug("UpdatePosition post postAction");

    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    // MidiManager.debug("UpdatePosition done");
  }
}
```

6.37 Volume.java

```
/**
 *
 */
package at.univie.csd.command;

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

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;
import org.nlogo.api.Syntax;

import at.univie.csd.Conversion;
import at.univie.csd.MidiCommand;

/**
 * @author Martin Dobiasch
 *
 */
public class Volume extends MidiCommand
{
  public Volume()
  {
    super();
  }

  public Syntax getSyntax()
  {
    return Syntax.commandSyntax(new int[] { Syntax.TYPE_NUMBER, Syntax.
        TYPE_NUMBER });
  }

  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {
    int channel;
    double vol;

    preAction();
    ShortMessage msg= new ShortMessage();
    try
    {
      channel= args[0].getIntValue();
      vol= args[1].getDoubleValue();
    }
    catch(LogoException e)
    {
      throw new ExtensionException(e.getMessage());
    }

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, 7, (int)
          Conversion.Rescale(vol, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
```

```
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    finally
    {
      postAction();
    }

    m_recv.send(msg, -1);

    postAction();
  }
}
```

6.38 Conversion.java

```
/**
 *
 */
package at.univie.csd;

/**
 * @author Martin Dobiasch (java portation), Erich Neuwirth
 * Helperclass for Conversion Tasks
 */
public class Conversion
{
  /**
   * Rescales a Number with given Lower and Upper Bound
   * @param x The Number to be rescaled
   * @param inmin Lower-Bound of the new number
   * @param inmax Upper-Bound of the new number
   * @return rescaled number
   */
  public static TwoByte TwoByteMidiRescale(double x, int inmin, int inmax)
  {
    long medval;
    TwoByte ret= new TwoByte();

    medval= Rescale(x, inmin, inmax, 0, 16383);
    ret.msb= (byte) (medval / (int) 128);
    ret.lsb= (byte) (medval % 128);

    return ret;
  }

  /**
   * Rescales a Number with given Lower and Upper Bound of the input
   *  and maxima for the output
   * @param x The Number to be rescaled
   * @param inmin Lower-Bound of the new number
   * @param inmax Upper-Bound of the new number
   * @param outmin Minimum of the Output
   * @param outmax Maximum of the Output
   * @return rescaled number
   */
```

```
  public static long Rescale(double x, int inmin, int inmax, int outmin, int
      outmax)
  {
    double result;

    result = ((x - inmin) / (inmax - inmin)) * (outmax - outmin) + outmin;

    if( result > outmax )
      result = outmax;
    if( result < outmin )
      result = outmin;

    return (long)result;
  }
}
```

6.39 Conductor.java

```
/**
 *
 */
package at.univie.csd.midi;

import java.util.ArrayList;

import org.nlogo.api.ExtensionException;


/**
 * @author Martin Dobiasch
 *
 */
public class Conductor
{
  public static final int NORMAL=  1;
  public static final int ENDLESS= 2;

  private Sheet[] sheets;
  private long[]  offs;
  private long starttime;
  private int playmode;

  public Conductor()
  {
    sheets= new Sheet[16];
    offs= new long[16];

    playmode= NORMAL;
  }

  /**
   * Provide access to the sheets of the conductor
   * @return sheets
   */
  public Sheet[] getSheets()
  {
```

```
  return sheets;
}

/**
 * Starts the conducting
 */
public synchronized void start()
{
  starttime= System.currentTimeMillis();

  for(int i= 0; i < 16; i++)
    offs[i]= 0;
}

/**
 * How much time has elapsed since start() had been performed
 * @return elapsed time since start
 */
public synchronized long elapsedTime()
{
  if( starttime == 0 )
    return 0;
  return System.currentTimeMillis() - starttime;
}

/**
 * Sets everything to start
 * starttime, and sheets are reset to start
 */
public synchronized void resettime()
{
  starttime= 0;

  for(int i= 0; i < 16; i++)
    offs[i]= 0;
}

/**
 * Remove all sheets
 */
public void clearSheets()
{
  for(int i= 0; i < 16; i++)
  {
    sheets[i]= null;
    offs[i]= 0;
  }
}

/**
 * Set all sheets to start
 */
public void resetSheets()
{
  int i;

  for(i= 0; i < 16; i++)
  {
    if( sheets[i] != null )
    {
```

```
      sheets[i].reset();
      offs[i]= 0;;

    }
  }
}

/**
 * Set Playmode
 * @param mode PlayMode
 */
public void setPlayMode(int mode)
{
  playmode= mode;
}

/**
 * Returns all events which are due.
 * Use this function after start() has been performed
 * @return due events in their order of sheets and occurence
 * @throws ExtensionException
 */
public ArrayList<TimedEvent> getDueEvents() throws ExtensionException
{
  long currTime;
  int i;
  ArrayList<TimedEvent> l= new ArrayList<TimedEvent>();
  TimedEvent ev;

  if( this.elapsedTime() == 0 )
  {
    this.start();
    currTime= 0;
  }
  else
    currTime= this.elapsedTime();

  for(i= 0; i < 16; i++)
  {
    if( sheets[i] != null )
    {
      ev= sheets[i].getCurrEvent();
      if( ev != null )
      {
          //throw new ExtensionException(ev.abstime + " " + currTime );
          if( ( offs[i] + ev.abstime ) <= currTime )
          {
            //send
            l.add(ev);
            sheets[i].NextEvent();
          }
      }
      else if( ( this.playmode & ENDLESS ) != 0)
      {
        sheets[i].reset();
        offs[i]+= sheets[i].getDuration();
      }
    }
  }
  return l;
```

```
  }

}
```

6.40 ConductorThread.java

```
/**
 *
 */
package at.univie.csd.midi;

import java.util.ArrayList;

import org.nlogo.api.Context;
import org.nlogo.api.ExtensionException;

import at.univie.csd.MidiManager;

/**
 * @author Martin Dobiasch
 *
 */
public class ConductorThread extends Thread
{
  private Context m_ctx;

  public ConductorThread(Context ctx)
  {
    super();
    m_ctx= ctx;
    // MidiManager.debug("CT init");

    this.setPriority(MIN_PRIORITY);
  }

  public void run()
  {
    Conductor c;
    try
    {
      // MidiManager.debug("CT1");
      while( true )
      {
        // MidiManager.debug("CT2");
        c= MidiManager.getMidiContext().getConductor();
        TimedEvent ev;
        ArrayList<TimedEvent> l;
        int i= 0;

        // MidiManager.debug("CT3");
        l= c.getDueEvents();
        // MidiManager.debug("CT4");

        // MidiManager.debug("Conductor conduct release Conductor " + l.size
            ());
        MidiManager.getMidiContext().releaseConductor();
```

```
          // MidiManager.debug("Conductor conduct start");

          for(i= 0; i < l.size(); i++)
          {
            ev= l.get(i);
            //throw new ExtensionException( ev.msg );

            //Runs a Command, and not waits for the command to terminate
            m_ctx.runCommand(ev.msg, false);
            // ctx.runCommand(ev.msg, true);

            //m_recv.send(ev.msg, ev.abstime);
          }

          // MidiManager.debug("Conductor conduct end");

          Thread.sleep(100);
          Thread.yield();
        }
      }
      catch( ExtensionException e)
      {
        // TODO:
        // MidiManager.debug("ExtensionException in CT");
        // MidiManager.debug(e.getMessage());
      }
      catch (InterruptedException e)
      {
        // TODO:
        // MidiManager.debug("InterruptedException in CT");
        // MidiManager.debug(e.getMessage());
      }
    }
}
```

6.41 Sheet.java

```
package at.univie.csd.midi;

import java.util.ArrayList;
import java.util.Iterator;

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

import org.nlogo.api.ExtensionException;

//import at.univie.csd.Conversion;
//import at.univie.csd.TwoByte;

/**
 * This is the class for the 'musicians'
 * Every sheet can contain several actions with a time code
 * @author Martin Dobiasch
 */
```

```
public class Sheet
{
  private ArrayList<TimedEvent> sEventQueue;
  private long abstime;
  private Iterator<TimedEvent> it;
  private TimedEvent curr;
  public final static long NO_TIMECODE= -1;

  public Sheet()
  {
    abstime= 0;
    sEventQueue= new ArrayList<TimedEvent>();
    curr= null;
    it= null;
  }

  /*private void addNote(double args[], long tc) throws ExtensionException
  {
    int channel;
    int note;
    double vol;
    int dur;
    ShortMessage msg1, msg2;

    msg1= new ShortMessage();
    msg2= new ShortMessage();

    try
    {
      channel= (int) args[0] - 1;
      note= (int) args[1];
      vol= args[2];
      dur= (int) args[3];

      msg1.setMessage(ShortMessage.NOTE_ON, channel, note, (int) Conversion
          .Rescale(vol, 0, 1, 0, 127));
      msg2.setMessage(ShortMessage.NOTE_OFF, channel, note, 0);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    catch(IndexOutOfBoundsException e)
    {
      throw new ExtensionException("not enough inputs for note " + e.
          getMessage());
    }

    TimedEvent ev= new TimedEvent();
    ev.abstime= abstime;
    if( tc != NO_TIMECODE )
      ev.abstime+= tc;
    ev.msg= msg1;
    sEventQueue.add(ev);

    ev= new TimedEvent();
    ev.abstime= abstime + dur - 1;
    ev.msg= msg2;
    sEventQueue.add(ev);
```

```
    if( tc == NO_TIMECODE )
      abstime+= dur;
    else
      abstime+= tc;
  }

  private void addNoteOn(double args[], long tc) throws ExtensionException
  {
    int channel;
    int note;
    double vol;
    ShortMessage msg1;

    msg1= new ShortMessage();

    try
    {
      channel= (int) args[0] - 1;
      note= (int) args[1];
      vol= args[2];

      msg1.setMessage(ShortMessage.NOTE_ON, channel, note, (int) Conversion
          .Rescale(vol, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    catch(IndexOutOfBoundsException e)
    {
      throw new ExtensionException("not enough inputs for noteon " + e.
          getMessage());
    }

    TimedEvent ev= new TimedEvent();
    ev.abstime= abstime;
    if( tc != NO_TIMECODE )
      ev.abstime+= tc;
    ev.msg= msg1;
    sEventQueue.add(ev);

    if( tc != NO_TIMECODE )
      abstime+= tc;
  }

  private void addNoteOff(double args[], long tc) throws ExtensionException
  {
    int channel;
    int note;
    ShortMessage msg1;

    msg1= new ShortMessage();

    try
    {
      channel= (int) args[0] - 1;
      note= (int) args[1];

      msg1.setMessage(ShortMessage.NOTE_OFF, channel, note, 0);
    }
```

```
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    catch(IndexOutOfBoundsException e)
    {
      throw new ExtensionException("not enough inputs for noteoff " + e.
          getMessage());
    }

    TimedEvent ev= new TimedEvent();
    ev.abstime= abstime;
    if( tc != NO_TIMECODE )
      ev.abstime+= tc;
    ev.msg= msg1;
    sEventQueue.add(ev);

    if( tc != NO_TIMECODE )
      abstime+= tc;
  }

  private void addKeypressure(double args[], long tc) throws
      ExtensionException
  {
    int channel;
    int note;
    double rval;
    ShortMessage msg;

    msg= new ShortMessage();

    try
    {
      channel= (int) args[0] - 1;
      note= (int) args[1];
      rval= args[2];

      msg.setMessage(ShortMessage.CHANNEL_PRESSURE, channel, note, (int)
          Conversion.Rescale(rval, 0, 1, 0, 127));
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    catch(IndexOutOfBoundsException e)
    {
      throw new ExtensionException("not enough inputs for keypressure " + e.
          getMessage());
    }

    TimedEvent ev= new TimedEvent();
    ev.abstime= abstime;
    if( tc != NO_TIMECODE )
      ev.abstime+= tc;
    ev.msg= msg;
    sEventQueue.add(ev);

    if( tc != NO_TIMECODE )
      abstime+= tc;
  }
```

```
private void addChannelPressure(double args[], long tc) throws
    ExtensionException
{
  int channel;
  double rval;
  ShortMessage msg;

  msg= new ShortMessage();

  try
  {
    channel= (int) args[0] - 1;
    rval= args[1];

    msg.setMessage(ShortMessage.CHANNEL_PRESSURE, channel, (int)
        Conversion.Rescale(rval, 0, 1, 0, 127), 0);
  }
  catch(InvalidMidiDataException e)
  {
    throw new ExtensionException("Invalid Midi data " + e.getMessage());
  }
  catch(IndexOutOfBoundsException e)
  {
    throw new ExtensionException("not enough inputs for noteon " + e.
        getMessage());
  }

  TimedEvent ev= new TimedEvent();
  ev.abstime= abstime;
  if( tc != NO_TIMECODE )
    ev.abstime+= tc;
  ev.msg= msg;
  sEventQueue.add(ev);

  if( tc != NO_TIMECODE )
    abstime+= tc;
}

private void addInstrument(double args[], long tc) throws
    ExtensionException
{
  int channel;
  int note;
  ShortMessage msg;

  msg= new ShortMessage();

  try
  {
    channel= (int) args[0] - 1;
    note= (int) args[1];

    msg.setMessage(ShortMessage.PROGRAM_CHANGE, channel, note - 1, 0);
  }
  catch(InvalidMidiDataException e)
  {
    throw new ExtensionException("Invalid Midi data " + e.getMessage());
  }
  catch(IndexOutOfBoundsException e)
```

```
    {
      throw new ExtensionException("not enough inputs for instrument " + e.
          getMessage());
    }

    TimedEvent ev= new TimedEvent();
    ev.abstime= abstime;
    if( tc != NO_TIMECODE )
      ev.abstime+= tc;
    ev.msg= msg;
    sEventQueue.add(ev);

    if( tc != NO_TIMECODE )
      abstime+= tc;
  }

  private void addPitchBend(double args[], long tc) throws
      ExtensionException
  {
    int channel;
    double rval;
    ShortMessage msg;
    TwoByte MyTwobyte = new TwoByte();

    msg= new ShortMessage();

    try
    {
      channel= (int) args[0] - 1;
      rval= args[1];

      MyTwobyte = Conversion.TwoByteMidiRescale(rval, -1, 1);

      msg.setMessage(ShortMessage.PITCH_BEND, channel, MyTwobyte.lsb,
          MyTwobyte.msb);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }
    catch(IndexOutOfBoundsException e)
    {
      throw new ExtensionException("not enough inputs for noteon " + e.
          getMessage());
    }

    TimedEvent ev= new TimedEvent();
    ev.abstime= abstime;
    if( tc != NO_TIMECODE )
      ev.abstime+= tc;
    ev.msg= msg;
    sEventQueue.add(ev);

    if( tc != NO_TIMECODE )
      abstime+= tc;
  }

  private void addController(long tc, int channel, int contr, double ival)
      throws ExtensionException
  {
```

```
    ShortMessage msg;

    msg= new ShortMessage();

    try
    {
      msg.setMessage(ShortMessage.CONTROL_CHANGE, channel - 1, contr, (int)
          ival);
    }
    catch(InvalidMidiDataException e)
    {
      throw new ExtensionException("Invalid Midi data " + e.getMessage());
    }

    TimedEvent ev= new TimedEvent();
    ev.abstime= abstime;
    if( tc != NO_TIMECODE )
      ev.abstime+= tc;
    ev.msg= msg;
    sEventQueue.add(ev);

    if( tc != NO_TIMECODE )
      abstime+= tc;
  }

  private void addMasterTuneFine(double args[], long tc) throws
      ExtensionException, IndexOutOfBoundsException
  {
    int channel;
    int lorpn;
    int hirpn;
    int lodata;
    int hidata;
    int cents;
    TwoByte tb;

    channel= (int) args[0];
    cents= (int) args[1];
    tb= Conversion.TwoByteMidiRescale(cents, -100, 100);

    lorpn= 1;
    hirpn= 0;
    lodata= tb.lsb;
    hidata= tb.msb;

    addRPN(channel, lorpn, hirpn, lodata, hidata, tc);
  }

  private void addRPN(int channel, int lorpn, int hirpn, int lodata, int
      hidata, long tc) throws ExtensionException
  {
    ShortMessage msg1, msg2, msg3, msg4, msg5, msg6;

    msg1= new ShortMessage();
    msg2= new ShortMessage();
    msg3= new ShortMessage();
    msg4= new ShortMessage();
    msg5= new ShortMessage();
    msg6= new ShortMessage();
```

```
channel--;

try
{
  msg1.setMessage(ShortMessage.CONTROL_CHANGE, channel, 101, (int)
      Conversion.Rescale(hirpn, 0, 1, 0, 127));
  msg2.setMessage(ShortMessage.CONTROL_CHANGE, channel, 100, (int)
      Conversion.Rescale(lorpn, 0, 1, 0, 127));
  msg3.setMessage(ShortMessage.CONTROL_CHANGE, channel,   6, (int)
      Conversion.Rescale(hidata, 0, 1, 0, 127));
  msg4.setMessage(ShortMessage.CONTROL_CHANGE, channel,  38, (int)
      Conversion.Rescale(lodata, 0, 1, 0, 127));
  msg5.setMessage(ShortMessage.CONTROL_CHANGE, channel, 101, 127);
  msg6.setMessage(ShortMessage.CONTROL_CHANGE, channel, 100, 127);
}
catch(InvalidMidiDataException e)
{
  throw new ExtensionException("Invalid Midi data " + e.getMessage());
}

TimedEvent ev= new TimedEvent();
ev.abstime= abstime;
if( tc != NO_TIMECODE )
  ev.abstime+= tc;
ev.msg= msg1;
sEventQueue.add(ev);

ev= new TimedEvent();
ev.abstime= abstime;
if( tc != NO_TIMECODE )
  ev.abstime+= tc;
ev.msg= msg2;
sEventQueue.add(ev);

ev= new TimedEvent();
ev.abstime= abstime;
if( tc != NO_TIMECODE )
  ev.abstime+= tc;
ev.msg= msg3;
sEventQueue.add(ev);

ev= new TimedEvent();
ev.abstime= abstime;
if( tc != NO_TIMECODE )
  ev.abstime+= tc;
ev.msg= msg4;
sEventQueue.add(ev);

ev= new TimedEvent();
ev.abstime= abstime;
if( tc != NO_TIMECODE )
  ev.abstime+= tc;
ev.msg= msg5;
sEventQueue.add(ev);

ev= new TimedEvent();
ev.abstime= abstime;
if( tc != NO_TIMECODE )
  ev.abstime+= tc;
ev.msg= msg6;
```

```
  sEventQueue.add(ev);

  if( tc != NO_TIMECODE )
    abstime+= tc;
}*/

public void addCommand(String command, long tc)
{
  TimedEvent ev;

  ev= new TimedEvent();
  ev.abstime= abstime;
  if( tc != NO_TIMECODE )
    ev.abstime+= tc;
  ev.msg= command;
  sEventQueue.add( ev );

  if( tc != NO_TIMECODE )
    abstime+= tc;
}

public void addCommand(String command, double[] args, long tc) throws
    ExtensionException
{
  /*try
  {
    if( command.equals("note") )
      addNote(args, tc);
    else if( command.equals ("noteon") )
      addNoteOn(args, tc);
    else if( command.equals("noteoff"))
      addNoteOff(args, tc);
    else if(command.equals("keypressure"))
      addKeypressure(args, tc);
    else if(command.equals("controller"))
      addController(tc, (int) args[0], (int) args[1], args[2]);
    else if(command.equals("instrument"))
      addInstrument(args, tc);
    else if(command.equals("channelpressure"))
      addChannelPressure(args, tc);
    else if(command.equals("pitchbend"))
      addPitchBend(args, tc);
    else if(command.equals("modulation"))
      addController(tc, (int) args[0], 1, Conversion.Rescale(args[1], 0,
          1, 0, 127) );
    else if(command.equals("volume"))
      addController(tc, (int) args[0], 7, Conversion.Rescale(args[1], 0,
          1, 0, 127) );
    else if(command.equals("expression"))
      addController(tc, (int) args[0], 11, Conversion.Rescale(args[1], 0,
          1, 0, 127) );
    else if(command.equals("pan"))
      addController(tc, (int) args[0], 10, Conversion.Rescale(args[1], -1,
          1, 0, 127));
    else if(command.equals("sustain"))
      addController(tc, (int) args[0], 64, Conversion.Rescale(args[1], 0,
          1, 0, 127) );
    else if(command.equals("reverb"))
      addController(tc, (int) args[0], 91, Conversion.Rescale(args[1], 0,
          1, 0, 127) );
```

```
      else if(command.equals("chorus"))
        addController(tc, (int) args[0], 93, Conversion.Rescale(args[1], 0,
            1, 0, 127) );
      else if(command.equals("resetcontrollers"))
        addController(tc, (int) args[0], 121, 0 );
      else if(command.equals("allnotesoff"))
        addController(tc, (int) args[0], 123, 0 );
      else if(command.equals("bankselectmsb"))
        addController(tc, (int) args[0], 32, args[1] );
      else if(command.equals("bankselectlsb"))
        addController(tc, (int) args[0], 0, args[1] );
      else if(command.equals("breath"))
        addController(tc, (int) args[0], 2, Conversion.Rescale(args[1], 0,
            1, 0, 127) );
      else if(command.equals("footpedal"))
        addController(tc, (int) args[0], 4, Conversion.Rescale(args[1], 0,
            1, 0, 127) );
      else if(command.equals("portamentotime"))
        addController(tc, (int) args[0], 5, args[1] );
      else if(command.equals("pitchbendrange"))
        addRPN((int) args[0], 0, 0, 0, (int) args[1], tc);
      else if(command.equals("mastertunecoarse"))
        addRPN((int) args[0], 2, 0, 0, (int) args[1] + 64, tc);
      else if(command.equals("mastertunefine"))
        addMasterTuneFine(args, tc);
      else
        throw new ExtensionException("Unknown command " + command );
    }
    catch( IndexOutOfBoundsException e )
    {
      throw new ExtensionException( "Not enough inputs to " + command + "\n"
          + e.getMessage() );
    }*/
  }

  public TimedEvent getCurrEvent() throws ExtensionException
  {
    if( curr == null )
    {
      if( it == null )
      {
        it= sEventQueue.iterator();
        curr= it.next();
        return curr;
      }
      else
      {
        if( it.hasNext() )
        {
          curr= it.next();
          return curr;
        }
        return null;
      }
    }
    else
      return curr;
  }

  public long getDuration()
```

```
  {
    return abstime;
  }

  /*public TimedEvent getNextEvent()
  {
    if( it == null )
      it= sEventQueue.iterator();

    return it.next();
  }*/

  public void NextEvent()
  {
    if( it != null )
    {
      if( it.hasNext() )
      {
        curr= it.next();
      }
      else
        curr= null;
    }
  }

  public void reset()
  {
    it= sEventQueue.iterator();
  }
}
```

6.42 TimedEvent.java

```
/**
 *
 */
package at.univie.csd.midi;

/**
 * @author Martin Dobiasch
 * Datatype for Events
 */
public class TimedEvent
{
  /**
   * Timecode of the event
   */
  public long   abstime;
  /**
   * Action to be performed
   */
  public String msg;
}
```

6.43 MidiCommand.java

```
package at.univie.csd;
/**
 * @author Martin Dobiasch
 */

import javax.sound.midi.MidiDevice;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.MidiUnavailableException;
import javax.sound.midi.Receiver;
import javax.sound.midi.Synthesizer;

import org.nlogo.api.Argument;
import org.nlogo.api.Context;
import org.nlogo.api.DefaultCommand;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.LogoException;

/**
 * @author Martin Dobiasch
 * Class for Commands for NetLogo using Midi
 * every Command before accessing the Midi-Device has to call preAction
    before
 * sending Commands to the device. After all Commands have been sent to the
 * device, postAction has to be called to unlock the Midi-Device again
 */
public class MidiCommand extends DefaultCommand
{
  private static MidiDevice m_dev;
  private static Synthesizer m_synth;
  protected static Receiver m_recv;


  public MidiCommand()
  {

  }

  /**
   * not used, has to be overwritten by of subclasses
   */
  public void perform(Argument[] args, Context ctx) throws
      ExtensionException,
      LogoException
  {

  }

  /**
   * Wait for the MidiContext, and lock it
   * @throws ExtensionException raised when Midi is unavailable
   */
  protected void preAction() throws ExtensionException
  {
    try
    {
      m_dev= MidiManager.getMidiContext().getDevice();
```

```
      if( !m_dev.isOpen() )
        m_dev.open();
      if( m_synth == null )
        m_synth = MidiSystem.getSynthesizer();
      if( !m_synth.isOpen() )
        m_synth.open();
      if( m_recv == null )
        m_recv = m_synth.getReceiver();
    }
    catch( MidiUnavailableException e)
    {
      throw new ExtensionException("Midi not available " + e.getMessage());
    }
    finally
    {
      MidiManager.getMidiContext().releaseDevice();
    }
  }

  /**
   * Release the MidiContext
   */
  protected void postAction()
  {
    //m_recv.close();
    //m_synth.close();
    //m_dev.close();
    MidiManager.getMidiContext().releaseDevice();
  }
}
```

6.44 MidiContext.java

```
/**
 *
 */
package at.univie.csd;

import javax.sound.midi.MidiDevice;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.MidiUnavailableException;

import org.nlogo.api.Context;

import at.univie.csd.midi.Conductor;
import at.univie.csd.midi.ConductorThread;

/**
 * @author Martin Dobiasch
 * This class takes care of the synchronized access to the midi device
 * it opens the midi device, and takes care of the Conductor
 */
public class MidiContext
{
  private boolean m_free;
  private boolean m_free_c;
```

```
private MidiDevice m_dev;
private Conductor m_cond;
private ConductorThread m_ct;

/**
 * Constructor
 * @throws MidiUnavailableException when midi is not available
 */
public MidiContext() throws MidiUnavailableException
{
  m_free= true;
  m_free_c= true;

  MidiDevice.Info[] infos = MidiSystem.getMidiDeviceInfo();
  m_dev = MidiSystem.getMidiDevice(infos[0]);

  m_cond= new Conductor();

  m_ct= null;
}

/**
 * Waits for the Device to be free, and locks it
 * @return Midi Device
 */
public synchronized MidiDevice getDevice()
{
  // if would be an error !!!
  while( !m_free )
  {
    try
    {
      wait();
    }
    catch(InterruptedException e)
    {
    }
  }
  m_free= false;

  return m_dev;
}

/**
 * Unlock the Device
 */
public synchronized void releaseDevice()
{
  m_free= true;
  notify();
}

/**
 * Waits for the Conductor to be free, and locks it
 * @return reference to the Conductor instance
 */
public synchronized Conductor getConductor()
{
  // if would be an error !!!
```

```
    while( !m_free_c )
    {
      try
      {
        wait();
      }
      catch(InterruptedException e)
      {
      }
    }
    m_free_c= false;
    return m_cond;
  }

  /**
   * Unlock the Conductor for other threads
   */
  public synchronized void releaseConductor()
  {
    m_free_c= true;
    notify();
  }

  public synchronized void startConductor(Context ctx)
  {
    if( m_ct != null ) //Conductor is running at the moment
    { //stop it
      m_ct.interrupt();
    }
    m_ct= new ConductorThread(ctx);
    m_ct.start();
  }

  public synchronized void stopCondcutor()
  {
    if( m_ct != null )
    {
      m_ct.interrupt();
      m_ct= null;
    }
  }
}
```

6.45 MidiManager.java

```
/**
 *
 */
package at.univie.csd;

import javax.sound.midi.MidiUnavailableException;

import org.nlogo.api.DefaultClassManager;
import org.nlogo.api.ExtensionException;
import org.nlogo.api.PrimitiveManager;
```

```
import at.univie.csd.command.AllNotesOff;
import at.univie.csd.command.ChannelPressure;
import at.univie.csd.command.Chorus;
import at.univie.csd.command.ConductorAddToSheet;
import at.univie.csd.command.ConductorAddToSheetWT;
import at.univie.csd.command.ConductorClearSheets;
import at.univie.csd.command.ConductorConduct;
import at.univie.csd.command.ConductorPlaymodeEndless;
import at.univie.csd.command.ConductorPlaymodeNormal;
import at.univie.csd.command.ConductorReset;
import at.univie.csd.command.ConductorStart;
import at.univie.csd.command.ConductorStop;
import at.univie.csd.command.Controller;
import at.univie.csd.command.Expression;
import at.univie.csd.command.Instrument;
import at.univie.csd.command.KeyPressure;
import at.univie.csd.command.MastertuneCoarse;
import at.univie.csd.command.MastertuneFine;
import at.univie.csd.command.Modulation;
import at.univie.csd.command.Note;
import at.univie.csd.command.NoteOff;
import at.univie.csd.command.NoteOn;
import at.univie.csd.command.Nrpn;
import at.univie.csd.command.Pan;
import at.univie.csd.command.Panic;
import at.univie.csd.command.PitchBend;
import at.univie.csd.command.PitchSens;
import at.univie.csd.command.Portamento;
import at.univie.csd.command.PortamentoFrom;
import at.univie.csd.command.PortamentoTime;
import at.univie.csd.command.ResetControllers;
import at.univie.csd.command.Reverb;
import at.univie.csd.command.Rpn;
import at.univie.csd.command.Sustain;
import at.univie.csd.command.UpdatePosition;
import at.univie.csd.command.Volume;

/**
 * @author Martin Dobiasch
 * Adds the Commands to NetLogo
 */
public class MidiManager extends DefaultClassManager
{
  private static MidiContext ctx;
  // private static BufferedWriter out;

  public MidiManager() throws ExtensionException
  {
    try
    {
      ctx= new MidiContext();
    }
    catch(MidiUnavailableException e)
    {
      throw new ExtensionException("Midi not available");
    }
    // openfile();
  }

  /*private void openfile()
```

```
{
  File file= new File("midiextensionlog.txt");
  try
  {
    out= new BufferedWriter(new FileWriter(file));
  }
  catch (IOException e)
  {
    //Debug.showMessage("const");
  }
}*/

/*public static void debug(String s)
{
  try
  {
    out.write(s + "\r\n");
    out.flush();
  }
  catch (IOException e)
  {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }
}*/

public static MidiContext getMidiContext()
{
  return ctx;
}

public void load(PrimitiveManager primitiveManager) throws
    ExtensionException
{
  primitiveManager.addPrimitive("noteon", new NoteOn());
  primitiveManager.addPrimitive("noteoff", new NoteOff());
  primitiveManager.addPrimitive("note", new Note());
  primitiveManager.addPrimitive("instrument", new Instrument());
  primitiveManager.addPrimitive("pitch.bend", new PitchBend());
  primitiveManager.addPrimitive("controller", new Controller());
  primitiveManager.addPrimitive("key.pressure", new KeyPressure());
  primitiveManager.addPrimitive("channel.pressure", new ChannelPressure())
      ;
  primitiveManager.addPrimitive("volume", new Volume());
  primitiveManager.addPrimitive("expression", new Expression());
  primitiveManager.addPrimitive("modulation", new Modulation());
  primitiveManager.addPrimitive("pan", new Pan());
  primitiveManager.addPrimitive("sustain", new Sustain());
  primitiveManager.addPrimitive("reverb", new Reverb());
  primitiveManager.addPrimitive("chorus", new Chorus());
  primitiveManager.addPrimitive("portamento.time", new PortamentoTime());
  primitiveManager.addPrimitive("portamento", new Portamento());
  primitiveManager.addPrimitive("portamento.from", new PortamentoFrom());
  primitiveManager.addPrimitive("rpn", new Rpn());
  primitiveManager.addPrimitive("nrpn", new Nrpn());
  primitiveManager.addPrimitive("reset.controllers", new ResetControllers
      ());
  primitiveManager.addPrimitive("all.notes.off", new AllNotesOff());
  primitiveManager.addPrimitive("pitch.sens", new PitchSens());
```

```
    primitiveManager.addPrimitive("mastertune.coarse", new MastertuneCoarse
        ());
    primitiveManager.addPrimitive("mastertune.fine", new MastertuneFine());
    primitiveManager.addPrimitive("panic", new Panic());

    primitiveManager.addPrimitive("updateposition", new UpdatePosition());

    primitiveManager.addPrimitive("conductor.clear.sheets", new
        ConductorClearSheets());
    primitiveManager.addPrimitive("conductor.add.to.sheet", new
        ConductorAddToSheet());
    primitiveManager.addPrimitive("conductor.add.to.sheet.list", new
        ConductorAddToSheetWT());
    primitiveManager.addPrimitive("conductor.restart", new ConductorReset())
        ;
    primitiveManager.addPrimitive("conductor.conduct", new ConductorConduct
        ());
    primitiveManager.addPrimitive("conductor.setplaymode.endless", new
        ConductorPlaymodeEndless() );
    primitiveManager.addPrimitive("conductor.setplaymode.normal", new
        ConductorPlaymodeNormal() );
    primitiveManager.addPrimitive("conductor.start", new ConductorStart() );
    primitiveManager.addPrimitive("conductor.stop", new ConductorStop() );
  }
}
```

6.46 TwoByte.java

```
/**
 *
 */
package at.univie.csd;

/**
 * Helper Datatype for MidiCSD
 * @author Martin Dobiasch (java portation), Erich Neuwirth
 *
 */
public class TwoByte
{
  public byte lsb;
  public byte msb;
}
```

Literatur

[1] Midi Manufacturer Assosiation. Gm 1 soundset. http://www.midi.org/techspecs/gm1sound.php. last seen: 18.11.2010.

[2] A Luedeke. Java-sound, midi. http://www-lehre.informatik.uni-osnabrueck.de/~aluedeke/java/javaSound/midi.html. last seen: 18.11.2010.

[3] E Neuwirth. Midicsd. http://sunsite.univie.ac.at/musicfun/MidiCSD/, 2008.

[4] Bomers F Pfisterer M. Java sound rescources: Faq: Midi programming. http://www.jsresources.org/faq_midi.html, 2000, 2005. last seen: 18.11.2010.

[5] U Schmidt. Oop mit java: Synchronisation von threads. http://www.fh-wedel.de/~si/vorlesungen/java/OOPMitJava/Multithreading/Synchronisation.html. last seen: 29.9.2010.

[6] U Wilensky. Netlogo. http://ccl.northwestern.edu/netlogo/, 1996.