Dokumentation UltraMixer MidiMap Editor

Inhaltsverzeichnis

1. Einführung

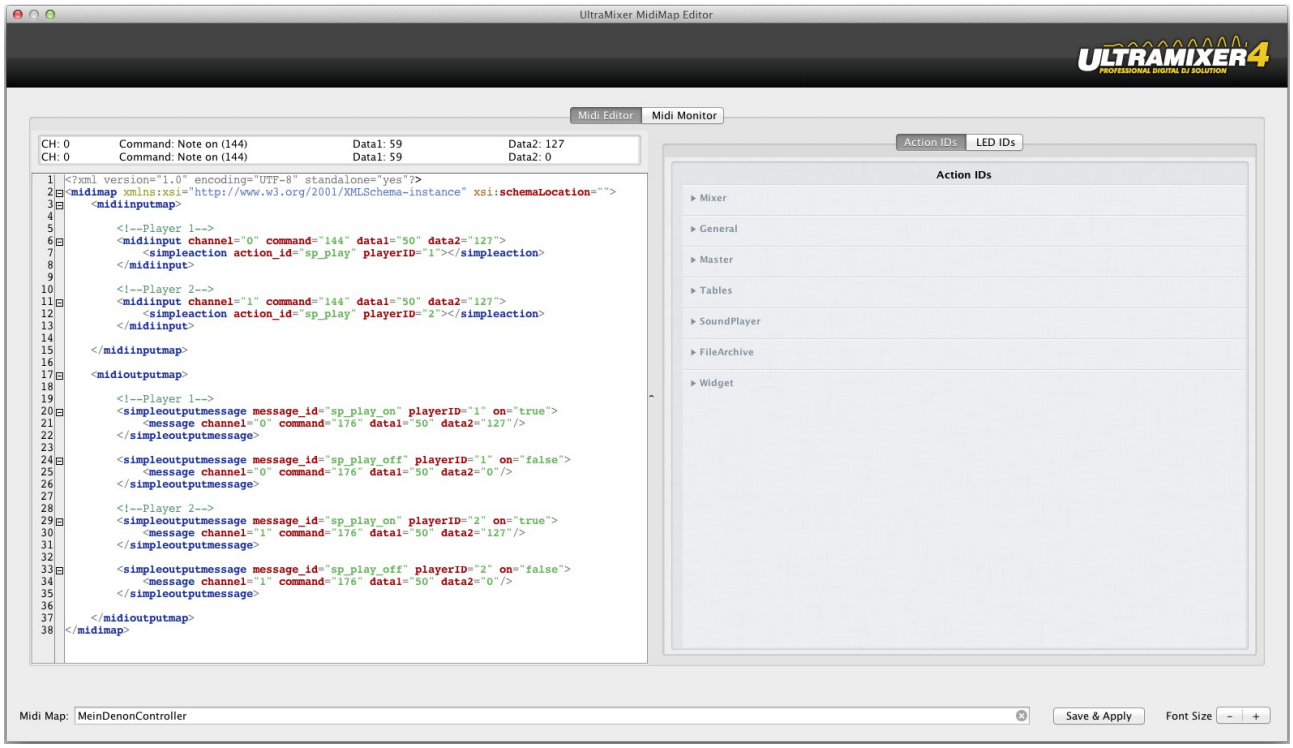
Auf Grund der Vielzahl an MIDI-Controllern auf dem Markt, können wir nur die bekanntesten und meistverkauften Controller im UltraMixer unterstützen. Damit aber auch Nutzer mit bisher nicht-unterstützten Controllern im UltraMixer arbeiten können, gibt es im UltraMixer 4 einen MidiMap Editor zum eigenständigen Mappen von Controllern.

Um einen MIDI-Controller erfolgreich zu mappen, benötigst Du einige Grundkenntnisse. Diese versuchen wir Dir nun in der folgenden Dokumentation zu vermitteln.

Zu finden ist der Editor im UltraMixer unter Einstellungen → MIDI/Remote. Du kommst nun in das MIDI-Auswahl-Fenster, wo Du im oberen Bereich unter „MIDI-Gerät“ deinen Controller auswählen kannst. Wenn Du nun in der darunter liegenden MIDI-Geräte-Liste Deinen Controller nicht findest, wird ein eigenes Mapping nötig. Klicke anschließend auf das Zahnrad-Symbol rechts oben und auf „Neu“, wenn Du ein komplett neues Mapping schreiben willst, oder auf „Bearbeiten“ um ein bereits ausgewähltes Mapping zu editieren. Anschließend öffnet sich der MidiMap Editor.

Grundsätzlich legst Du nun in einer XML-Datei fest, welche Taste Deines Controllers, welche Funktion im UltraMixer auslöst. Dabei hat jede Taste Deines Controllers einen MIDI-Signal-Wert und jede Funktion im UltraMixer eine ActionID.

Du siehst nun nach dem Öffnen des Editors folgendes Grundgerüst Deiner XML-Datei mit einigen Beispielevents:



Hier kannst Du nach dem midimap-Tag außerdem einen info-Tag einfügen der dir die Versionierung erleichtert:

```
<info midiVersion="1" type=„xmllcls“>
    <author>UltraMixer</author>
    <date>2014/01/23</date>
    <manufacturer>Denon</manufacturer>
    <productname>MC 2000</productname>
</info>
```

Gib Deiner MIDI-Map unten links einen logischen Namen und klicke auf den Button „Save & Apply“. Diesen Button musst Du nun immer betätigen um eine Veränderung in Deiner XML-Datei zu testen.

Hinweis: Um sich am Ende besser in Deiner erstellten XML-Datei zurecht zu finden, kommentiere so viele Funktionen wie möglich in dieser Art:

<!--{Text}-->

Es gibt nun 6 unterschiedliche Arten von Events auf Deinem Controller:

- normale Klick-Events, welche durch Betätigen einer Taste ausgelöst werden
- Drehregler und Schieberegler-Events mit Anfang und Ende
- Drehregler-Events ohne Anfang und Ende
- Events, welche über eine Tastenkombination mit einer Shift-Taste zu erreichen sind
- Jogwheel-Events
- LED-Licht-Events

Wenn Du eine Taste auf Deinem Controller betätigst, findest Du oben links im MIDI Monitor folgende Werte, welche Du für die XML benötigst: Channel, Command, Data1, Data2.

Achte dabei darauf, dass bei den meisten Tasten der Wert „Data2“ und manchmal auch „Data1“ und „Command“ beim Betätigen und Loslassen der Taste unterschiedlich ist.

Achte außerdem besonders darauf, dass Du den richtigen Channel in Deinem MIDI-Input-Event einträgst, da dieser von Controller zu Controller sehr variiert!

Auf der rechten Seite findest Du die ActionID für die Funktion im UltraMixer. Wenn Du also zum Beispiel die ID für die Taste Play suchst klicke rechts auf die Gruppe SoundPlayer. Nun findest Du unter Play die ID „sp_play“.

Beachte, dass es viele Tasten auf Deinem Controller 2 mal gibt, einmal für Player 1 und einmal für Player 2. Die unterschiedlichen Tasten besitzen natürlich auch unterschiedliche MIDI-Werte. Hat der Regler oder eine Taste keine explizite Playerzugehörigkeit, kann die „playerID“ auch weggelassen werden!

Nun folgen konkrete Anwendungsbeispiele bei der Eintragung der Werte in die XML-Datei. Halt Dir dabei immer vor Augen, dass es sich ausschließlich um Beispiele handelt. Die MIDI-Werte sind von Controller zu Controller komplett unterschiedlich.

2. MIDI-Input-Mapping

2.1 Klick-Events

Beispielwerte:

Player 1:

CH: 0, Command: Note on (144), Data1: 59, Data2: 127

Player 2:

CH: 0, Command: Note on (144), Data1: 66, Data2: 127

ID: sp_play

→

Player 1:

```
<midiinput channel="0" command="144" data1="59" data2="127">
    <simpleaction actionID="sp_play" playerID="1"></simpleaction>
</midiinput>
```

Player 2:

```
<midiinput channel="0" command="144" data1="66" data2="127">
    <simpleaction actionID="sp_play" playerID="2"></simpleaction>
</midiinput>
```

Die „actionID“ und die „playerID“ wird innerhalb der „simpleaction“ eingetragen. Im „midiinput“ werden channel, command, data1 und data2 eingetragen.

2.2 Drehregler/Schieberegler-Events mit Anfang und Ende

Das Attribut „relative“ muss auf „false“ stehen wenn bei „data2“ absolute Werte geliefert werden. Also zum Beispiel je nach Stellung des Reglers Werte zwischen 1 und 127. Sollte je nur ein Werte für Vorwärts und Rückwärts geliefert werden muss das Attribut auf „true“ stehen. Lässt man es weg, steht es generell auf „false“. Das Attribut „relativeFactor“ bestimmt die Schnelligkeit eines Reglers. Dabei sind Werte zwischen 0.0 und 1.0 möglich. 0.0 ist dabei normale Geschwindigkeit und 1.0 die größte mögliche Beschleunigung.

Beispielwerte:

Player 1:

CH: 0, Command: Control Change (176), Data1: 20

Player 2:

CH: 0, Command: Control Change (176), Data1: 21

ID: eq_low_player

→

Player 1:

```
<midiinput channel="0" command="176" data1="20" data2="-1" hasMinjorIncrement="false">
    <simpleaction actionID="eq_low_player" playerID="1" directionForward="true" relative=“false“
        relativeFactor=“0.0“></simpleaction>
</midiinput>
```

Player 2:

```
<midiinput channel="0" command="176" data1="21" data2="-1" hasMinjorIncrement="false">
    <simpleaction actionID="eq_low_player" playerID="2" directionForward="true" relative=“false“
        relativeFactor=“0.0“></simpleaction>
</midiinput>
```

*Sollte der Regler nun falsch herum implementiert sein, muss „directionForward“ auf „false“ gestellt werden. Werden bei einem Regler abhängig von dessen Standpunkt 2 verschiedene data1-Werte geliefert, zum Beispiel Data1: 18 & 19, sieht der XML-Teil zu diesem **einen** Regler folgendermaßen aus:*

Player 1:

```
<midiinput channel="0" command="176" data1="18" data2="-1" hasMinjorIncrement="true">
    <simpleaction actionID="eq_low_player" playerID="1" directionForward="true">
    </simpleaction>
</midiinput>
<midiinput channel="0" command="176" data1="19" data2="-1"    hasMinjorIncrement="false">
    <simpleaction actionID="eq_low_player" playerID="1" directionForward="true">
    </simpleaction>
</midiinput>
```

Dabei muss also darauf geachtet werden, bei dem ersten data1-Wert „hasMinjorIncrement“ auf „true“ zu stellen. Dieser Parameter wird oftmals bei 14-Bit-Controllern benötigt, da diese mehr Werte bei einem Regler liefern als herkömmliche Controller.

2.3 Drehregler-Events ohne Anfang und Ende

Beispielwerte:

Player 1:

nach Links:

CH: 0, Command: Control Change (176), Data1: 27, Data2: 127

nach Rechts:

CH: 0, Command: Control Change (176), Data1: 27, Data2: 1

Player 2:

nach Links:

CH: 0, Command: Control Change (176), Data1: 30, Data2: 127

nach Rechts:

CH: 0, Command: Control Change (176), Data1: 30, Data2: 1

ID nach links: fx_flanger_player_relative_backward

ID nach rechts: fx_flanger_player_relative_forward

→

Player 1:

```
<midiinput channel="0" command="176" data1="27" data2="127">
    <simpleaction actionID="fx_flanger_player_relative_backward" playerID="1"></simpleaction>
</midiinput>
<midiinput channel="0" command="176" data1="27" data2="1">
    <simpleaction actionID="fx_flanger_player_relative_forward" playerID="1"></simpleaction>
</midiinput>
```

Player 2:

```
<midiinput channel="0" command="176" data1="30" data2="127">
    <simpleaction actionID="fx_flanger_player_relative_backward" playerID="2"></simpleaction>
</midiinput>
<midiinput channel="0" command="176" data1="30" data2="1">
    <simpleaction actionID="fx_flanger_player_relative_forward" playerID="2"></simpleaction>
</midiinput>
```

2.4 Shift-Events

Beispielwerte:

Player 1:

Shift:

CH: 0, Command: Note on (144), Data1: 97, Data2: 127 / 0

Taste:

CH: 0, Command: Note on (144), Data1: 99, Data2: 127

Player 2:

Shift:

CH: 0, Command: Note on (144), Data1: 98, Data2: 127 / 0

Taste:

CH: 0, Command: Note on (144), Data1: 100, Data2: 127

ID Loop verdoppeln: sp_loop_increase

ID Loop halbieren: sp_loop_decrease

→

Player 1:

```
<midiinput channel="0" command="144" data1="97" data2="127">
        <conditionchangeaction stateCount="2" state="0" actionID="ShiftP1" playerID="1"/>
</midiinput>
<midiinput channel="0" command="144" data1="97" data2="0">
        <conditionchangeaction stateCount="2" state="1" actionID="ShiftP1" playerID="1"/>
</midiinput>
```

Wird die Shift-Taste auch noch für andere Aktionen benötigt, reicht die einmalige Deklaration, da sie über die in dem Fall frei wählbare „actionID“ (im Beispiel “ShiftP1“) eindeutig erreichbar ist. Außerdem muss auf die Anzahl der verschiedenen Stati (stateCount) und die eindeutige Bezeichnung des Status (state, wird von 0 hochgezählt) geachtet werden.

```
<midiinput channel="0" command="144" data1="99" data2="127">
        <conditionalaction actionID="LoopUpDownP1" playerID="1">
            <actions>
                <simpleaction actionID="sp_loop_increase" playerID="1"></simpleaction>
                <simpleaction actionID="sp_loop_decrease" playerID="1"></simpleaction>
            </actions>
            <conditionchangeaction stateCount="2" actionID="ShiftP1" playerID="1">
            </conditionchangeaction>
        </conditionalaction>
</midiinput>
```

Zu sehen ist jetzt eine „conditionalaction“ welche die „simpleactions“ und die zugehörige „conditionchangeaction“, welche auch als Bedingung umschrieben werden könnte, enthält.

*Dabei ist die obere „simpleaction“ jene, welche **mit** Betätigung der Shift-Taste ausgeführt wird, die untere ist jene, welche **ohne** Betätigung der Shift-Taste ausgeführt wird. Die wurde in der Deklaration der Shift-Taste festgelegt. Die „actionID“ der „conditionalaction“ kann dabei frei gewählt werden, die „actionID“ der „conditonchangeaction“ muss allerdings der „actionID“ der Shift-Tasten-Deklaration entsprechen.*

Player 2:

```
<midiinput channel="0" command="144" data1="98" data2="127">
        <conditionchangeaction stateCount="2" state="0" actionID="ShiftP2" playerID="2"/>
</midiinput>
<midiinput channel="0" command="144" data1="98" data2="0">
        <conditionchangeaction stateCount="2" state="1" actionID="ShiftP2" playerID="2"/>
</midiinput>
<midiinput channel="0" command="144" data1="100" data2="127">
      <conditionalaction actionID="LoopUpDownP2" playerID="2">
          <actions>
             <simpleaction actionID="sp_loop_increase" playerID="2"></simpleaction>
             <simpleaction actionID="sp_loop_decrease" playerID="2"></simpleaction>
          </actions>
          <conditionchangeaction stateCount="2" actionID="ShiftP2" playerID="2">
          </conditionchangeaction>
      </conditionalaction>
</midiinput>
```

2.5 Jogwheel-Events

2.5.1 Berührungsunempfindliche Jogwheels

Achtung! Werden beim Jogwheel für links und rechts nur je ein Wert im MIDI-Monitor angezeigt und die Schnelligkeit der Rotation vernachlässigt, verhält sich das Jogwheel wie ein Drehregler ohne Anfang und Ende.

Auf Grund des sehr umfangreichen Beispiels, haben wir in dem Fall darauf verzichtet das Mapping für beide Player darzustellen. Grundsätzlich gibt es abgesehen von den MIDI-Werten zwischen den Playern bzw. Decks auch keine Unterschiede.

Beispielwerte:

Jogwheel:

CH: 0, Command: Control Change (176), Data1: 25

Umschalter Scratching/Pitchbending:

CH: 0, Command: Note on (144), Data1: 72, Data2: 127

ID Scratchen: sp_jogwheel_scratching

ID Pitchpending: sp_jogwheel_pb

→

Zu allererst die Deklaration des Umschalters Scratching/Pitchbending:

```
<midiinput channel="0" command="144" data1="72" data2="127">
    <conditionchangeaction stateCount="2" actionID="Vinyl_Mode_P1" playerID="1">
    </conditionchangeaction>
</midiinput>
```

Nun wird abhängig von der Einstellung des Umschalters eine Aktion ähnlich einer Shift-Action ausgeführt:

```
<midiinput channel="0" command="176" data1="24" data2="-1">
   <conditionalaction actionID="Scratch_P1" playerID="1" directionForward="true">
      <actions>
```

```
        <simpleaction actionID="sp_jogwheel_scratching" playerID="1"  rotaryMaxValue="100"
      absoluteToRelative="false" swapSpeed="false" swapData1Data2="false"></simpleaction>

        <simpleaction actionID="sp_jogwheel_pb" playerID="1" rotaryMaxValue="50"
      absoluteToRelative="false" swapSpeed="false" swapData1Data2="false"></simpleaction>

    </actions>

    <conditionchangeaction stateCount="2" actionID="Vinyl_Mode_P1" playerID="1">

    </conditionchangeaction>

  </conditionalaction>

</midiinput>
```

Mit dem Parameter „rotaryMaxValue“ kann man einstellen, wie empfindlich der UltraMixer auf ein Scratchen und Pitchbenden reagiert. Hier muss man einfach austesten bei welchem Wert das Scratchen am besten klingt. Falls die Richtung vertauscht ist, kann man auch hier den Parameter „directionForward“ auf „false“ stellen.

Werden bei „data2“ Positionswerte statt Schnelligkeitswerten geliefert, hilft es einen Parameter „absoluteToRelative“ in den midiinput-tag einzuführen und auf „true“ zu stellen.

Außerdem kann man die data1- und data2-Werte vertauschen indem man swapData1Data2 auf „true“ stellt. Auch dies ist in seltenen Fällen nötig.

Ein weiteres mögliches Attribut ist „swapSpeed“. Dies ist eine Variable um die Logik zu drehen, weil einige Controller für Vorwärts 127 senden, andere wiederum 65. Das Attribut muss also bei Controllern, welche Bewegungswerte von um die 65 liefern auf „true“ gesetzt werden.

2.5.2 Berührungsempfindliche Jogwheels

Beispielwerte:

Jogwheel:

CH: 0, Command: Control Change (176), Data1: 25

Umschalter Scratching/Pitchbending:

CH: 0, Command: Note on (144), Data1: 72, Data2: 127

Touchpad:

CH: 0, Command: Note on (144), Data1: 77, Data2: 127 / 0

ID Scratching an: sp_jogwheel_scratching_on

ID Scratching aus: sp_jogwheel_scratching_off

In diesem Fall wird es etwas komplizierter, da ein weiterer MIDI-Wert für das Berührungs-Touchpad auf dem Jogwheel beachtet werden muss.

→

```
<midiinput channel="0" command="144" data1="72" data2="127">
   <conditionchangeaction stateCount="2" actionID="Vinyl_Mode_P1" playerID="1">
   </conditionchangeaction>
</midiinput>
```

Zu allererst wurde wieder der Umschalter Scratching/Pitchbending deklariert. Es folgt eine doppelt verschachtelte „conditionalaction“.

```
<midiinput channel="0" command="176" data1="25" data2="-1">
   <conditionalaction actionID="Scratch_Search_JW_P1" playerID="1" directionForward="false">
            <actions>
               <conditionalaction actionID="Scratch_JW1" playerID="1">
                  <actions>
```

```
                    <simpleaction actionID="sp_jogwheel_scratching" playerID="1"
                     rotaryMaxValue="100" jogwheelTouchable="true"></simpleaction>

                    <simpleaction actionID="sp_jogwheel_scratching" playerID="1"
                     rotaryMaxValue="100" jogwheelTouchable="false"></simpleaction>

                </actions>

                <conditionchangeaction stateCount="2" actionID="ScratchP1" playerID="1" >

                </conditionchangeaction>

            </conditionalaction>

            <simpleaction actionID="sp_jogwheel_pb" playerID="1" rotaryMaxValue="50">

            </simpleaction>

        </actions>

        <conditionchangeaction stateCount="2" actionID="Vinyl_Mode_P1" playerID="1">

        </conditionchangeaction>

    </conditionalaction>

</midiinput>
```

Das Attribut „jogwheelTouchable“ muss dabei auf „true“ stehen, damit das berührungsempfindliche Jogwheel anhält und während des Scratchens nicht weiter läuft, falls das Jogwheel grade nicht gedreht wird.

```
<midiinput channel="0" command="144" data1="77" data2="127">

    <conditionalaction actionID="JW_T_ON" playerID="1">

        <actions>

            <simpleaction actionID="sp_jogwheel_scratching_on" playerID="1"></simpleaction>

            <simpleaction></simpleaction>

        </actions>

        <conditionchangeaction stateCount="2" actionID="Vinyl_Mode_P1" playerID="1">

        </conditionchangeaction>
```

```
    </conditionalaction>
</midiinput>
<midiinput channel="0" command="144" data1="77" data2="0">
    <conditionalaction actionID="JW_T_OFF" playerID="1">
        <actions>
            <simpleaction actionID="sp_jogwheel_scratching_off" playerID="1">
            </simpleaction>
            <simpleaction></simpleaction>
        </actions>
        <conditionchangeaction stateCount="2" actionID="Vinyl_Mode_P1" playerID="1">
        </conditionchangeaction>
    </conditionalaction>
</midiinput>
```

Der vorangegangene Code bewirkt nun, dass der UltraMixer weiß, wann mit Scratchen angefangen und aufgehört wurde um dann sofort den Song weiterspielen zu lassen. Ist der Scratch-Modus also aktiviert wird bei Berüherung des Jogwheels die „actionID“ „sp_jogwheel_scratching_on“ ausgeführt, im Pitchbend-Modus bleibt die „simpleaction“ leer. Das gleiche Prozedere folgt beim loslassen des berührungsempfindlichen Jogwheels mit der „actionID“ namens „sp_jogwheel_scratching_off“.

2.6 Übersicht möglicher Tags und Attribute

Haupt-Tag:

<midiinput>

mögliche Attribute: channel, command, data1, data2

weitere Tags:

<conditionchangeaction>

mögliche Attribute: actionID, playerID, stateCount, state

<conditionalaction>

mögliche Attribute: actionID, playerID, directionForward, swapSpeed

<simpleaction>

mögliche Attribute: absoluteToRelative, hasMinjorIncrement, ignoredData2, actionID, playerID, rotaryMaxValue, jogwheelTouchable, directionForward, relative, relativeFactor, swapSpeed, swapData2Data2

<lockactionlist>

mögliche Attribute: -

<actions>

mögliche Attribute: -

3. MIDI-Output-Mapping

3.1 LED-Licht-Events

Anders als bei den vorangegangenen Events gehören die Licht-Ereignisse nicht in die MIDI-Input-Map, sondern in die MIDI-Output-Map. Die zugehörigen IDs findet man rechts oben unter „LED IDs“.

Beispielwerte:

Player 1:

CH: 0, Command: Note on (144), Data1: 51, Data2: 127 / 0

Player 2:

CH: 0, Command: Note on (144), Data1: 60, Data2: 127 / 0

ID an: sp_cue_on

ID aus: sp_cue_off

→

Player 1:

```
<simpleoutputmessage messageID="sp_cue_on" playerID="1" on="true">
        <message channel="0" command="144" data1="59" data2="127"/>
</simpleoutputmessage>
<simpleoutputmessage messageID="sp_cue_off" playerID="1" on="false">
        <message channel="0" command="144" data1="59" data2="0"/>
</simpleoutputmessage>
```

Player 2:

```
<simpleoutputmessage messageID="sp_cue_on" playerID="2" on="true">
        <message channel="0" command="144" data1="66" data2="127"/>
</simpleoutputmessage>
<simpleoutputmessage messageID="sp_cue_off" playerID="2" on="false">
        <message channel="0" command="144" data1="66" data2="0"/>
</simpleoutputmessage>
```

Im Unterschied zur MIDI-Input-Map gibt es in der MIDI-Output-Map noch den Parameter „on“, mit dem man steuert wann die LED an und aus ist.

Die MIDI-Output-Werte können dabei auch komplett unterschiedlich zu den Input-Werten sein. In diesem Fall musst Du in das Datenblatt des Controllers schauen oder die Werte schlicht im in den Editor integrierten MIDI-Simulator testen. Dabei kannst Du bei den Parametern channel, data1 und data2 statt einem konkreten Wert einen Wertebereich angeben. Zum Beispiel: data1=“50-60“.

Auch im Output-Bereich gibt es die Möglichkeit Bedingungen zu definieren, beziehungsweise Shift-Events zu erzeugen. Dies sieht in etwa so aus:

```
<conditionchangeoutputmessage stateCount="2" messageID="Shift_P1" playerID="1">
    <state stateNumber="0">
        <message channel="1" command="176" data1="21" data2="0"/>
    </state>
    <state stateNumber="1">
        <message channel="1" command="176" data1="21" data2="127"/>
    </state>
    <conditionaloutputmessage stateNumber="0" messageID="Flanger_Del" playerID="1">
        <simpleoutputmessage messageID="fx_flanger_player_onoff_on" playerID="1" on="true">
            <message channel="1" command="176" data1="11" data2="127"/>
        </simpleoutputmessage>
        <simpleoutputmessage messageID="fx_flanger_player_onoff_off" playerID="1" on="false">
            <message channel="1" command="176" data1="11" data2="0"/>
```

```
        </simpleoutputmessage>
    </conditionaloutputmessage>
    <conditionaloutputmessage stateNumber="0" messageID="CutOff_Cue1" playerID="1">
        <simpleoutputmessage messageID="fx_cutoff_player_onoff_on" playerID="1" on="true">
            <message channel="1" command="176" data1="12" data2="127"/>
        </simpleoutputmessage>
        <simpleoutputmessage messageID="fx_cutoff_player_onoff_off" playerID="1" on="false">
            <message channel="1" command="176" data1="12" data2="0"/>
        </simpleoutputmessage>
    </conditionaloutputmessage>
</conditionchangeoutputmessage>
```

3.2 Lichtsteuerung über MIDI

Es ist außerdem möglich, Licht-Events über die manuelle MIDI-Auswahl zu steuern. Dazu ist das vorgehen ähnlich wie bei dem Mappen eines Controllers.

Man wählt unter Einstellungen → Steuerung „other MIDI device“ in der „MIDI-Gerät“ - Liste aus. Nun wählt man unter „Manuelle MIDI-Auswahl“ das gewünschte „MIDI-Gerät“ der Lichtsteuerung aus.

Nun drückt man Menu → New und ist im MIDI Editor.

Die MIDI-Input-Map kann nun vernachlässigt werden, wichtig ist nur die MIDI-Output-Map.

So könnte ein mögliches Mapping aussehen:

```
<midimap xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="">
<info version="1">
    <author>UltraMixer</author>
    <date>2014/01/23</date>
</info>
  <midioutputmap>
    <!--Player 1-->
    <simpleoutputmessage messageID="sp_play_on" playerID="1" on="true">
      <message channel="0" command="176" data1="100" data2="127"/>
    </simpleoutputmessage>
    <simpleoutputmessage messageID="sp_play_off" playerID="1" on="false">
      <message channel="0" command="176" data1="100" data2="0"/>
    </simpleoutputmessage>
    <simpleoutputmessage messageID="sp_beattick_on" playerID="1" on="true">
      <message channel="0" command="176" data1="111" data2="127"/>
    </simpleoutputmessage>
```

```
    <simpleoutputmessage messageID="sp_beattick_off" playerID="1" on="false">
      <message channel="0" command="176" data1="111" data2="0"/>
    </simpleoutputmessage>
     <!--Player 2-->
    <simpleoutputmessage messageID="sp_play_on" playerID="2" on="true">
      <message channel="0" command="176" data1="102" data2="127"/>
    </simpleoutputmessage>
    <simpleoutputmessage messageID="sp_play_off" playerID="2" on="false">
      <message channel="0" command="176" data1="102" data2="0"/>
    </simpleoutputmessage>
     <simpleoutputmessage messageID="sp_beattick_on" playerID="2" on="true">
      <message channel="0" command="176" data1="112" data2="127"/>
    </simpleoutputmessage>
    <simpleoutputmessage messageID="sp_beattick_off" playerID="2" on="false">
      <message channel="0" command="176" data1="112" data2="0"/>
    </simpleoutputmessage>
  </midioutputmap>
</midimap>
```

Im vorangegangenen Code wird ein MIDI-Out-Signal bei „Play“, „Stop“, „Beat“ und „kein Beat“ gesendet. Die zugehörigen IDs findet man wie auch beim Mapping eines Controllers rechts unter „LED IDs“. Die „message“ ist dabei frei wählbar, das bedeutet channel, command, data1 und data2 kann man frei belegen, muss aber mit den MIDI-Input-Werten des Empfangsgerätes beziehungsweise den Einstellungen in der genutzten Licht-Software überein stimmen.

3.3 Weitere MIDI-Geräte anschließen

Um weitere MIDI-Geräte anzuschließen musst Du wieder in das MIDI-Auswahl-Fenster unter Einstellungen → MIDI/Remote gehen. Hier kannst Du links unten mit einem Klick auf das Plus-Symbol beliebig viele MIDI-Geräte hinzufügen. Über das danach erscheinende Minus-Symbol kannst Du Geräte natürlich auch wieder entfernen.

Es ist nun also möglich Licht und beliebig viele Controller über MIDI zu steuern.