Documentation UltraMixer MidiMap Editor

Contents

1. Introduction

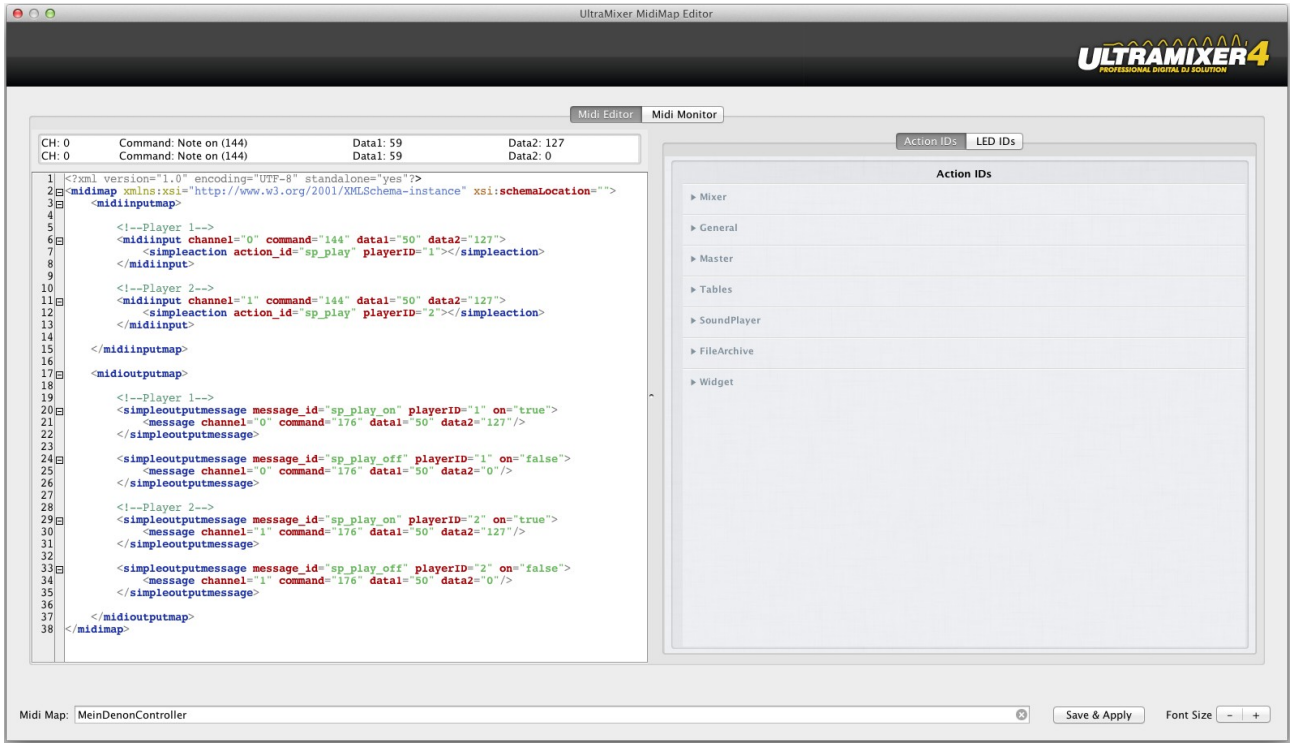
Due to the variety of MIDI controllers on the market, we can only support the best-known and best-selling controllers in UltraMixer. In UltraMixer 4 there is a MidiMap editor for users with previously unsupported controllers for independent mapping of these controllers.

To map a MIDI controller successfully, you need some basic knowledge. We are now trying to convey you this knowledge in the following documentation.

In UltraMixer you find the editor under Preferences → MIDI / Remote. Now you see the MIDI selection window where you can select your controller in the "MIDI Device" section at the top. If you do not find your controller in the underlying „MIDI Mapping“ list, a selfmade mapping is necessary. Therefor you have to click on the gear icon on the top right. Now choose "New" if you want to write a completely new mapping, or take "Edit" to edit an already selected mapping. Afterwards, the MidiMap editor opens.

Fundamentally you determine which button of your controller triggers which function in UltraMixer. For that you need a XML file. Each button of your controller has a MIDI signal value and each function in UltraMixer an actionID.

After opening the editor you see the following basic structure of your XML file with some sample events:



After the midimap-tag you also could insert an info-tag for easier versioning:

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

Give your MIDI-map a logical name at the bottom left and click on the button "Save & Apply". You always have to press this button to test a change in your XML file.

Note: In order to find your way in your XML file you have created, annotate as many features as possible in this way:

<!--{Text}-->

There are six different types of events on your controller:

- normal click events that are triggered by pressing a button
- rotary control and slider events with a beginning and end
- rotary contro events without a beginning or end
- events, which are accessible via a key combination with a shift key
- jogwheel events
- LED light events

If you press a button on your controller, you'll find top left of the MIDI Monitor the following values, which you need for your XML file: Channel, Command, Data1, Data2.

Pay attention that the value "Data2" and maybe also „Data1“ or „Command“ is different while pressing and releasing the key.

Pay also special attention to enter the correct channel in your MIDI input event, because it varies from controller to controller very much!

On the right side you will find the „ActionID“ for the function in UltraMixer. For example if you look for the ID of the play button, you will find it right by clicking on the group sound player. Now you can take the ID "sp_play", which you find under „Play“.

Pay attention, that every button on your controller could be available 2 times, once for player 1 and once for Player 2. The several keys also have different MIDI values. If the controller button have no explicit player affiliation, the playerID can also be left out!

It follows concrete examples for the incorporation of values in the XML file. Watch out that the grey frames are only examples. The MIDI-values are completely different from controller to controller.

2. MIDI input mapping

2.1 Click events

Example values:

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>
```

The "actionID" and the "playerID" have to be entered within the "simpleaction". In the "midiinput" - part you have to enter channel, command, data1 and data2.

2.2 Rotary control/slider control events with start and end

*The attribute "relative" has to be set to "false" when absolute values are provided in „data2“. So, for example, depending on the position of the controller values between 1 and 127. If only one value each for forward and reverse are supplied, the attribute hast to be set to "true". If you leave it off, it is generally set to "false". The "relative factor" attribute determines the speed of a control. Values between 0.0 and 1.0 are possible. 0.0 is normal speed, 1.0 the greatest possible acceleratio*n.

Example values:

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>
```

If the control is implemented in the wrong direction now, you have to set "directionForward" to "false". If there are two different data1 values depending on the position of the control, for example Data1: 18 & 19, the XML part of this control looks like this:

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>
```

In this case you have to ensure, that the parameter "hasMinjorIncrement" in the first midiinput-event is "true". This parameter is often required for 14-bit controllers, because these provide more values for a control than a conventional controller.

2.3 Rotary control events without start and end

Example values:

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

Example values:

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>
```

If you need the shift key also for other actions, a unique declaration is enough, because of the freely selectable "actionID" (in this example "ShiftP1"), which is clearly reachable. Anyway you have to take care about the number of different states (stateCount) and the unique identification of the states (state, is incremented from 0).

```
<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>
```

You can see a "conditionalaction" which contains the "simpleactions" and the associated "conditionchangeaction", which could be described as a condition.

*The upper "simpleaction" is executed **with** pressing the shift key, the lower "simpleaction" is executed **without** pressing the shift key. The "actionID" of the "conditionalaction" can be chosen freely, the "actionID" of the "conditonchangeaction" have to correspond to the "actionID" of the shift key declaration.*

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 Touch-insensitive jogwheels

Attention! If you only see one value for jogwheel left and one value for jogwheel right in the MIDI monitor and the speed of rotation is neglected, the jogwheel acts like a rotary control without a beginning or end.

Due to the very large example, we only show you the mapping of one deck which is for both players represent however. Basically, there are no differences between the players and decks except from their MIDI values.

Example values:

Jogwheel:

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

Scratching/pitchbending switch:

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

ID scratching: sp_jogwheel_scratching

ID pitchpending: sp_jogwheel_pb

→

First of all, the declaration of the scratching/pitchbending switch:

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

Now an action, which depends on the setting of the switch, similar to a shift action, is executed:

```
<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">
        </simpleaction>

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

        </simpleaction>

      </actions>

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

      </conditionchangeaction>

   </conditionalaction>

</midiinput>
```

With the parameter "rotaryMaxValue" you can adjust how sensitive the UltraMixer scratches and pitchbends. Here you just have to test the value, where the scratching sounds best. If the direction is reversed, you can put the "directionForward" parameter to "false" here.

If there are positions values for "data2" instead of speed values, the parameter "absoluteToRelative" helps. Just introduce it into the midiinput-tag and set it to "true".

In addition, you can swap the data1 and data2 values by setting „swapData1Data2 to "true" represents. This is necessary in rare cases.

Another attribute is "swapSpeed". This is a variable to rotate the logic, because some controllers send a 127 for forward, others 65. So the attribute has to be set to „true“ for controllers, which send values of around 65.

2.5.2 Touch-sensitive jogwheels

Example values:

Jogwheel:

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

Scratching/pitchbending switch:

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

Touchpad:

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

ID scratching on: sp_jogwheel_scratching_on

ID scratching off: sp_jogwheel_scratching_off

In this case it is more complicated, because you have to be aware of another MIDI value for the touchpad on the jogwheel.

→

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

First of all, the scratching/pitchbending switch is declared again. The following is a double nested "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="true"></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>
```

The attribute "jogwheelTouchable" has to be set to "true", so the song stops while touching the touch-sensitive jogwheel and does not continue during the scratching, if the jogwheel is not rotated right now.

```
<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>
```

Because of the preceding code the UltraMixer know when scratching was started and stopped. So it continues playing the song immediately. If the scratch mode is activated, the "actionID" "sp_jogwheel_scratching_on“ is executed. In the pitchbend mode the „simpleaction“ has to be empty. The same procedure follows for the release of the touch-sensitive jogwheel with the "actionID" called "sp_jogwheel_scratching_off".

2.6 Overview of possible tags and attributes

main tag:

<midiinput>

possible attributes: channel, command, data1, data2

additional tags:

<conditionchangeaction>

possible attributes: actionID, playerID, stateCount, state

<conditionalaction>

possible attributes: actionID, playerID, directionForward, swapSpeed

<simpleaction>

possible attributes: absoluteToRelative, hasMinjorIncrement, ignoredData2, actionID, playerID, rotaryMaxValue, jogwheelTouchable, directionForward, relative, relativeFactor, swapSpeed, swapData2Data2

<lockactionlist>

possible attributes: -

<actions>

possible attributes: -

3. MIDI output mapping

3.1 LED light events

In contrast to the previous events, the light events not belong to the midiinputmap, but rather to the midioutputmap. The corresponding IDs can be found under "LED IDs" on the top right.

Example values:

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>
```

In contrast to the midiinputmap, in the midioutputmap there is a parameter "on", which controls when the LED is on and off.

The MIDI output values could be completely different from the input values. In this case you have to look in the datasheet of the controller or you simply have to test the values in the integrated MIDI simulator. Here you can place a range of values for the parameters channel, data1 and data2 instead of a concrete value. For example: data1 = "50-60".

In the Output section there is a possibility to define conditions or to generate shift events, too. This looks something like this:

```
<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 Light control via MIDI

It is also possible to control light events by the manually MIDI selection. The proceed is similar to the mapping of a controller.

You select Settings → Control "other MIDI device" in the "MIDI device" - list. Now you select "Manual MIDI selection" and select the "MIDI device" of the light control.

Now press Menu → New and you are in the MIDI Editor.

The MIDI input map can now be neglected, it is the MIDI-output map what matters.

A possible mapping could look like this:

```
<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>
```

In the previous code a MIDI out signal will be sent on "Play", "Stop", "beat" and "no Beat". You find the corresponding IDs as well as in mapping a controller right under "LED IDs". The "message" is freely selectable, which means channel, command, data1 and data2 can be freely assigned, but the MIDI input values must match to the receiving device or the settings in the used light software.

3.3 Connect additional MIDI devices

To connect additional MIDI devices you have to select Settings → MIDI again. In this window you can add as many MIDI devices as you want by clicking on the plus button left at the bottom. If you add an device there will appear a minus-button, where you can delete a device, of course.

Now it is possible to control light and any number of controllers via MIDI.