SuperConductor

Users Guide

What is *SuperConductor*?

Instead of being the technician, be the musician. Let *SuperConductor* remix your entire set in an instant with total recall of all MIDI mapped parameters at a single pedal press. It tweaks fader levels, adjusts reverb sends, turns on and off devices, fires off clips - anything that can be MIDI mapped is instantly recalled. Later in the song, fire off a melody to multiple instruments while cranking the gain and compression on your synth input with the next pedal push. SuperConductor reworks your entire set on the fly so you can keep playing the music.

Setup

SuperConductor is an Ableton Live MIDI Device implemented using Max for Live (M4L). M4L using at least Max version 7 is required to use this device. Installation is easy: after downloading

and expanding the zip archive you'll find two items. Before opening Live, copy the **song-cues.json text file** anywhere into Max's search path, for example *Users/Shared/Max 7/Library***.** Then, open your Live set and simply drag and drop the **SuperConductor.amxd device** into a new MIDI track.

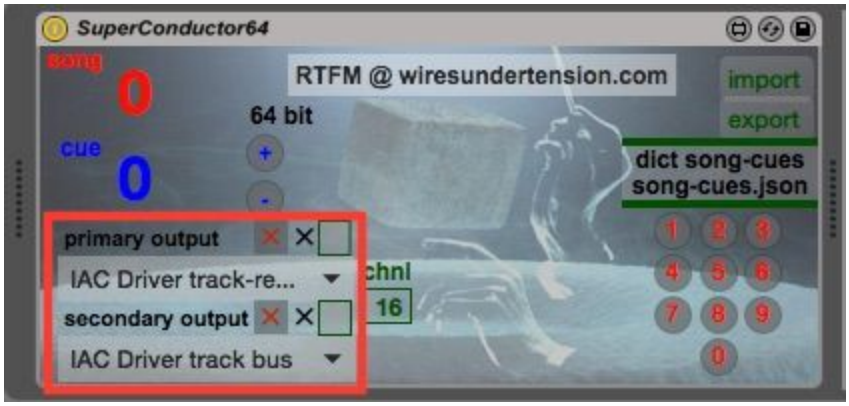
SuperConductor sends remote MIDI messages, including continuous controller messages (which Live does not normally output from MIDI tracks!) using the power of virtual MIDI busses[1]. On Mac, this is trivial using the built-in IAC bus. On Windows, several free third-party options are available. First, follow this guide to setup two virtual MIDI busses for use in Ableton Live. In Ableton Live Preferences > MIDI Sync, enable these busses for both **"Track" and "Remote" Input** and **"Track" Output**.

SuperConductor neither receives nor outputs MIDI through the standard device chain (it outputs MIDI internally) so input and output on its MIDI track could be disabled if desired.

SuperConductor uses standard MIDI messages and not the Live Object Model for two reasons:

1. The Live Object Model does not support access to a very large number of devices and settings. In contrast, almost *everything* is MIDI mappable in Live.
2. We don't need to run these messages at audio rate, but we do need them to happen "instantly". MIDI bussing will be faster than running messages through the Max4Live bridge using the Live Object Model.

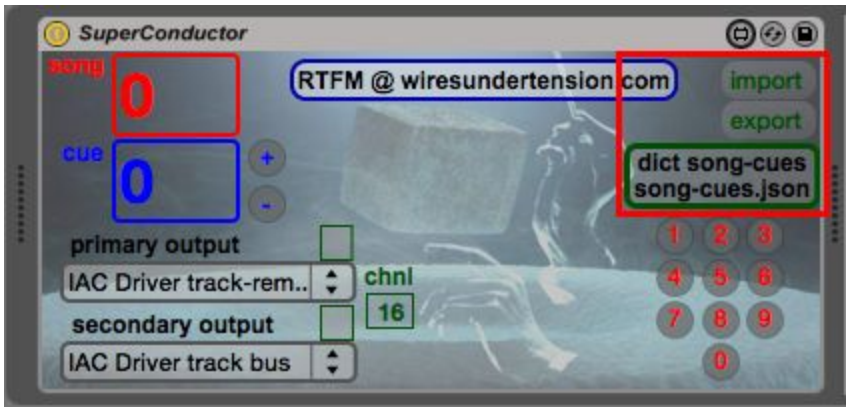
MIDI routing



In the highlighted **pulldown menus** above, you can choose MIDI outputs for the note and controller messages SuperConductor will send. SuperConductor supports sending these messages out to up to two outputs. Choose one of the virtual MIDI busses for the primary output and the other for the secondary output. When a valid port is selected a **black "X"** appears above each pulldown menu. Also above each pulldown menu is a **red toggle box** that enables each output. Make sure these are checked to send MIDI data to the respective outputs.

[1] Thanks be to Daivd Butler for his 64-bit **imp.midi** objects to make this possible.

Designing Cues



SuperConductor allows you to organize songs into a sequence of numbered cues. Each cue sends a collection of MIDI continuous controller and / or note messages out either the primary or secondary outputs, or both. Using a virtual MIDI bus that has **Remote** enabled input allows these messages to control MIDI mappings. Output can be used to play instruments on MIDI tracks that read from these virtual MIDI busses. Cues are stored in the **dict song-cues** object highlighted above which loads the **song-cues.json** JSON file on startup. Double-click this object to open the editor:

```
{
      "1-0" : [ "0 1 127 1", "0 2 127 1 2000", "0 3 127 1 0 2" ],
      "1-1" : [ "0 1 0 1", "0 2 0 1 1000", "0 3 0 1 0 3" ],
      "2-0" : [ "1 64 64 500 1", "1 66 64 500 1 500", "1 67 64 500 2 1000 1" ]
}
```

The JSON dictionary above uses a “<song>-<cue>” format for keys. Above we have cue entries for song one / cue zero, song one / cue one, and song two / cue zero. Cues, which follow the colons, are arrays of string tuples, one per message, surrounded with double quotes and separated by commas. Each tuple begins with either a **0 for a continuous controller message** or a **1 for a note message**. The remaining tuple format for each type is:

Continuous Controller (type 0):
“0 <*controller number*> <*controller value*> <*MIDI channel*> [<*delay ms*> <*routing*>]”

Note (type 1):
“1 <*pitch*> <*velocity*> <*duration ms*> <*MIDI channel*> [<*delay ms*> <*routing*>]”

Above, note that the **<delay ms>** and **<routing>** fields are optional. The **<delay ms>** value indicates the number of milliseconds that the message should be delayed by before sending. The **<routing>** value can be one of **1**, **2**, or **3** and indicates whether the message should be

sent out the primary output only (1), the secondary output only (2) or both outputs (3). When omitted the message is only sent out the primary output.

For note values, if <duration ms> = 0, only a note-on message will be sent. Beware stuck notes (these can be silenced by hitting the transport stop button)!

In the example above, the first cue ("1-0") triggers three continuous controller messages, for controller numbers 1, 2, and 3, all with value 127. These could be used to turn on MIDI mapped device toggles for example. The second message is delayed by 2000 milliseconds. All values are sent out MIDI channel 1. The first two are sent out the primary output by default, and the last is sent out the secondary output (<routing> = 2).

The second cue ("1-1") sets these same controllers to zero (e.g. turning off the same devices) with a 1000 millisecond delay for controller two. Also, note that controller 3's message is sent to both primary and secondary outputs (<routing> = 3).

The last cue ("2-0") plays C, D, E♭ with a velocity of 64, separated by 500 milliseconds. Each note also has a duration of 500 milliseconds. The final E♭ is sent out MIDI channel 2 of the primary output. Instruments on MIDI tracks that read from these channels will receive these notes.

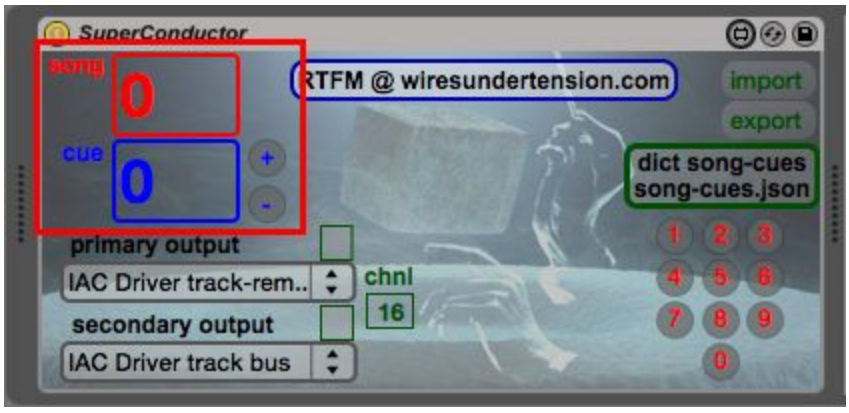
The dictionary editor enforces a strict format in these entries to produce valid JSON. All entries in the cue array must be separated by commas excluding the last entry. Likewise, note the comma following the first array, separating entries in the dictionary. All entries are separated by commas excluding the last entry. If you make a mistake in the format the editor will alert you to the line where the error is detected before letting you close and loading your changes into the dictionary.

The dictionary editor's capabilities are quite limited. For example, there is no undo-redo support. For elaborate changes, consider copying the entire contents into a real text editor, making the required changes and then replacing the entire contents back in the dictionary editor.

In order to preserve your changes for next time, you need to overwrite the song-cues.json text file on disk. This is done by clicking the **export** button highlighted above. **Note that saving your Live set WILL NOT save changes you've made to your song-cues dictionary. You need to export song-cues files manually following any changes.**

The **import** button above will let you read in any valid JSON dictionary file, replacing the contents of the dictionary. This is useful for keeping alternate versions of your song-cues in different files, for example as a backup of previous ideas. Also, if you change the dictionary using the editor, don't like the changes and would like to revert to the saved version, simply re-import the file.

Activating Cues

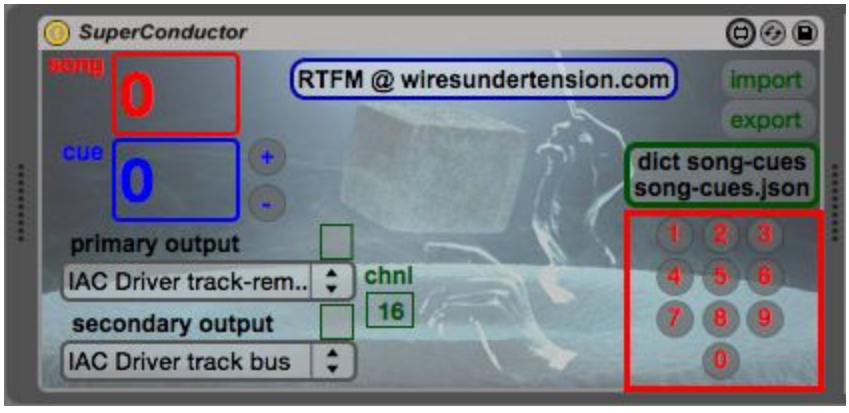


Cues are triggered using the **song** and **cue** number boxes highlighted above. Setting a value in the **song** number box automatically resets **cue** to 0 and fires off the corresponding cue. It also will stop the Live set from playing which sends an “All Notes Off” MIDI message, silencing any hanging note-on messages.

Setting a value for **cue** actives the cue corresponding to the displayed <song-cue> entry. If the dictionary contains an entry for the cue, the background of these number boxes turns from transparent to white.

Cues can also be incremented and decremented using the mappable Live buttons to the right of the **cue** number box. Map MIDI pedals to these **+** and **-** buttons to activate cues on the fly while keeping your hands free to play music.

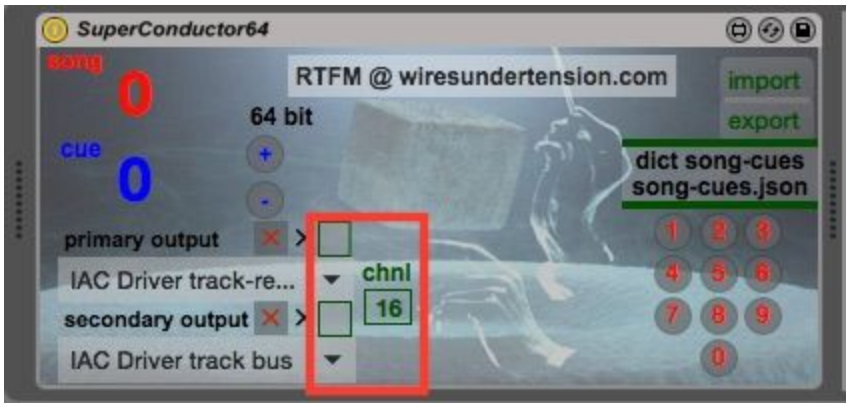
Song Hot Buttons



The **numbered mappable buttons** highlighted above are used to reset the **song** number box to each of the 10 labeled values. Consider assigning these to MIDI controller buttons or the

corresponding computer number keys for an easy way to reset your Live set for the start of each song.

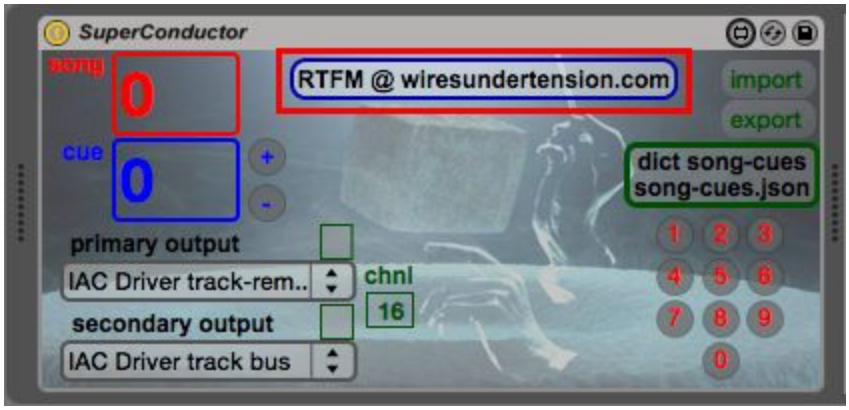
Song-Cue Output



The <song>-<cue> selected can itself be used to trigger a continuous controller message with controller number <song> and value <cue> sent out one of the device outputs. This is useful for sending <song>-<cue> information to other **wiresundertension.com** devices like *TriggerFinger* that receive these mssages to lookup their own <song>-<cue> data.

The green **toggle buttons** highlighted above indicate which of the outputs will pass along this controller message. The **channel number box** indicates on what MIDI channel this message will be sent.

Find this Manual



With an active internet connection, this manual can be accessed anytime by clicking the button highlighted above. A browser will open and direct you to the wiresundertension.com page for this guide.