

MIDI to Web-API User Manual

Software and manual version 0.9, June 2018, © Sebastian Beutel, s.beutel@avolites.de

This software is published under the New BSD license. See LICENSE.txt for more details.

Special thanks go to Alex del Bondio for ideas, testing, and brilliant feedback.

This software was designed to be useful in conjunction with Avolites lighting consoles. However, it was not made by Avolites Ltd., London, nor can or shall Avolites Ltd. be held responsible or liable for anything regarding this software.

Purpose and how it works

This software translates MIDI messages – USB-MIDI from class compliant devices – to http-requests which can be predefined in an easy way, can contain the MIDI level (velocity), and are then sent to a user-defined IP address for further processing. The purpose is to translate MIDI commands to Avolites Titan Web API commands so that you can control your Titan installation by your favorite MIDI gadget.

Prerequisites

In order to use this program you need:

- An Avolites Titan Mobile, installed and running, Titan v10 or newer. This software doesn't work with Titan One (this is not allowed – and technically the Titan One doesn't feature the Web API) nor with standalone consoles (USB MIDI devices are not detected there). Titan Simulator allows for Web API usage from v11 on.
- The Midityweb software which needs to run on a Windows® computer. Usually this will be the computer you run Titan (i.e. the software for your Titan Mobile) on. It can, however, also be a separate computer which is networked with your Titan Mobile or standalone Titan console. In that case set the correct target IP in the config.ini.
- A class-compliant MIDI device, connected to your computer. There are various smaller and larger controllers on the market, e.g. the Behringer BCF-2000, the Korg Nanokontrol, the AKAI APC 25 or 40, and various others.
- Edit the config.ini file to suit your needs. This is done with a simple text editor, and due to the plaintext format and inline documentation it shouldn't be too difficult.

Quick start

If installed using the installer then you'll find the software in the Avolites program group. Launch it from there.

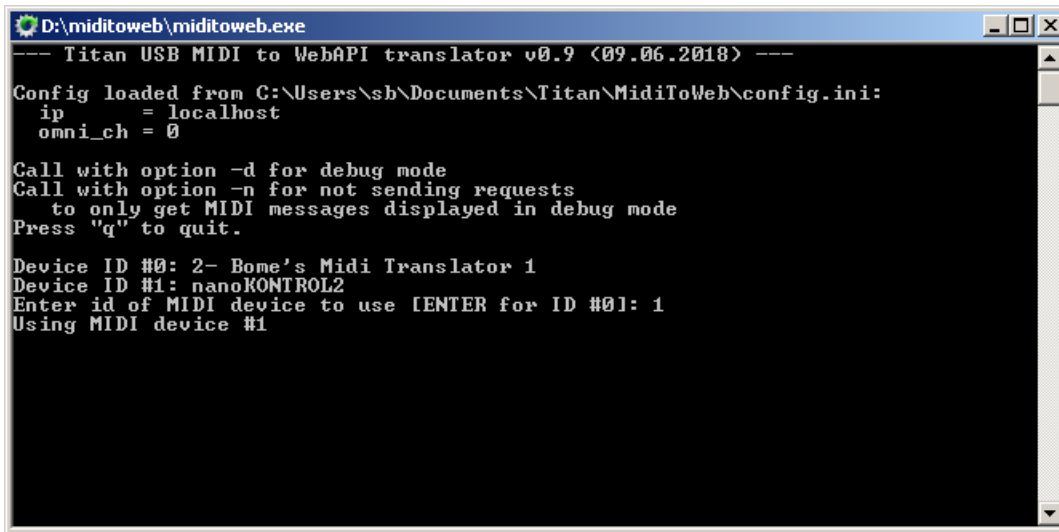
If not installed but simply unzipped then start the software by double-clicking midityweb.exe.

The config.ini (see next section) is read upon software start. Apply any changes there before you launch the program.

The software finds available MIDI devices and shows a list to choose from. Enter the ID of the device you are going to use followed by <ENTER> to make the software start listening to this device.

You might minimize the program window (a console window). In order to close the software, either press <q>, or close it with the [X] button in the window's title bar.

Miditoweb.exe starts as command line program in a command prompt:



```
D:\miditoweb\miditoweb.exe
--- Titan USB MIDI to WebAPI translator v0.9 (09.06.2018) ---
Config loaded from C:\Users\sb\Documents\Titan\MidiToWeb\config.ini:
ip = localhost
omni_ch = 0
Call with option -d for debug mode
Call with option -n for not sending requests
to only get MIDI messages displayed in debug mode
Press "q" to quit.
Device ID #0: 2- Bome's Midi Translator 1
Device ID #1: nanoKONTROL2
Enter id of MIDI device to use [ENTER for ID #0]: 1
Using MIDI device #1
```

Here, we got two MIDI devices (Bomes Midi Translator and a nanoKONTROL), entered <1> <ENTER> to select the nanoKONTROL, the software confirms to use device #1, and can now be minimized and run in the background.

Options

Miditoweb.exe can also be started with either of two options:

- Option -d (thus called miditoweb.exe -d) starts the debug mode which displays detailed information about the received MIDI commands as well as the mapped (and sent) API http request.
- Option -n (miditoweb.exe -n) starts the no request mode. This is useful in conjunction with the -d option (start the program like this: miditoweb.exe -d -n) if you simply want to display MIDI and API information without actually sending requests (and possibly waiting for an answer which never comes). You can easily find out which MIDI commands your device is sending out this way.

See section "Finding out MIDI messages" on p. 6 for some examples.

The config.ini

Miditoweb does not run without, or with a corrupt, config file. You might consider having various files, for special applications or different devices (e.g. some with faders, others only with buttons). Simply rename/replace the config.ini in that case.

If the program was installed using the installer then most likely the config.ini from "My Documents"\Titan\MidiToWeb is used – you can easily open it for editing. Another config.ini will be in your program folder ("Program Files"\Avolites\MiToWeb) – this config is a fallback and only used if the other one is not available. This mechanism is to allow editing without the need for administrator privileges.

When the program starts it displays the full path from where the config was loaded:

```
Config loaded from C:\Users\sb\Documents\Titan\MidiToWeb\config.ini:
```

N.B. if you install the program over an existing installation then your config.ini will be kept, and a new file config.ini.latest will be created. In order to use this, copy/rename it to 'config.ini'.

If the program was not installed but just the zip archive downloaded and unpacked then the config.ini from this folder will be used. In that case the program only displays the relative path:

```
Config loaded from config.ini:
```

Of course you may manually copy this file to "My Documents"\Titan\MidiToWeb in order to make use of the a.m. mechanism. It is recommended that you always have an original config.ini with the program, just in case editing it goes wrong.

The config.ini is a simple text file. Blank lines, as well as everything after a semi colon (;) in a line, is ignored. You are encouraged to use lines starting with ; as comments, to remind you what you did before.

Values are defined in a key = value syntax. There are three sections:

[general]

Here are the general settings:

- `ip` – this is the IP address or hostname where the http requests shall be sent to. The default value is localhost, assuming you run this software on your Titan Mobile computer. For other network configurations set the IP address according to your network. N.B.: all requests go to port 4430 which is the specific port for the Titan Web API. This is hardcoded into miditoweb.exe
- `omni_ch` – by default this is 0 which makes the software follow the MIDI channel of each message and map the command string only if the channel matches. However you can make the software behaving like in omni mode by setting another value: e.g. if you set `omni_ch = 5` then each incoming message, ignoring its channel, is mapped to commands as like as it was sent with channel = 5.

[strings]

The [strings] section holds a collection of useful command strings which can be referenced by their number.

The command strings are derived from the Avolites Web API documentation at <http://api.avolites.com/11.0/>.

All the strings mentioned here have been tested with Titan Web API v11.0. They may or may not work with other versions of the API. In particular they are likely to fail when trying to use Titan v10.0. In that case, (1) update your Titan installation to v11. If this is not an option then (2) try to figure the correct strings from the Web API documentation at <http://api.avolites.com/10.0/>.

Optionally the placeholders \$arg and \$lev can be used in the strings:

- `$arg` is an argument which can be inserted when the command string is mapped to a particular MIDI message. This way you can use the same string to flash Playback #1 with a MIDI message, and Playback #27 with another message – just use \$arg as playback number, and set the correct numbers when mapping this to message A or B. Be aware that the argument can only be an integer value.
- `$lev` is the normalized level sent with your MIDI message, to be used in the mapped string. MIDI sends levels in 7bit values = 0...127. Miditoweb.exe then converts this to a value 0.0...1.0 which is accepted by the Web API interface

Some examples for commonly used command strings are already included in the config ini:

```
1 = titan/script/2/Playbacks/FirePlaybackAtLevel?  
   handle_userNumber=$arg&level_level=$lev&alwaysRefire=false
```

String no. 1 might be the most common command. The function [FirePlaybackAtLevel\(\)](#) sets a particular playback's level, and expects the arguments `handle` – which is here passed as user number - , `level`, and 'always refire' as Boolean value. For normal use we will pass the `userNumber` when the string is mapped (thus we write `= $\$arg$`), and supply the level received via MIDI (hence `= $\$lev$`).

```
2 = titan/script/2/Playbacks/SwopPlayback?handle_userNumber=$arg
```

String no. 2 swops a particular playback using [SwopPlayback\(\)](#). Note that this function doesn't require a level which is why we don't use `$\$lev$` here. However we still use `$\$arg$` to pass our handle user number.

```
3 = titan/script/2/Playbacks/ClearSwopPlayback?handle_userNumber=$arg
```

String no. 3 clears the previously swopped playback with our user number which we again pass as `$\$arg$` . Make yourself familiar with the fact that MIDI messages and API commands are commands to change a state. And since we have previously swopped this playback, it will stay swopped until we 'unswop' it using this command. It is highly recommended to map `Swop/Flash` commands to `NoteOn` messages, and to map `ClearSwop/ClearFlash` commands to the `NoteOff` messages of the same MIDI notes. This makes sure (if you haven't completely messed up your controller's settings) that pressing a key swops/flashs, and releasing this key clears the swop/flash.

```
10 = titan/script/2/Masters/SetMaster?handle_titanId=$arg&value=$lev
```

This is similar to string no.1 except that this time we have a function which requires a Titan ID that we now pass as argument. See the next section for details on the Titan IDs.

In general, you can identify items in three ways:

- By `handle_userNumber` (see e.g. string #1) – this allows to e.g. create some mapping with MIDI faders assigned to playbacks with user numbers 901~910. Now, in Titan, simply give the handles you want to control by MIDI the correct user number, and you are good to go.
- By `handle_titanId` (see e.g. string #10) – this is most commonly used for masters, as there is reason to suspect they have frequently the same titan IDs (and as there is no suitable other command available)
- By `handle_location` – this way you could e.g. assign some faders or flash buttons to page 2 buttons 1~10 of the playbacks window. Then, in Titan simply move some cues there to make them react to MIDI.

Have a look at section [Web API commands](#), [Titan IDs](#) and [Locations](#) on p. 8 to learn more about identifiers.

There might be cases where you need neither a level nor an argument – just omit `$\$arg$` and `$\lev` (and set 0 as argument in the mapping section).

There might be cases where you'd rather pass strings as arguments. This is not supported. But then, be creative, and make a bunch of strings. E.g. if you want to reference handles by their location then the full parameter would be something like `handle_location=Playbacks_2_1` where 'playback' refers to the main playback fader handles, 2 is the page number, and 1 is the playback index on this page. Instead of attempting to use `Playbacks_2_1` as `$\$arg$` – which will fail – simply create a string with `...Playbacks_2_ $\$arg$...` (this will do for page 2, the playback index to be passed as argument), `...Playbacks_3_ $\$arg$...` (same for page 3) and so on:

```
21 = titan/script/2/Playbacks/FirePlaybackAtLevel?
    handle_location=PlaybackWindow_1_$arg&level_level=$lev&alwaysRefire=false

22 = titan/script/2/Playbacks/FirePlaybackAtLevel?
    handle_location=PlaybackWindow_2_$arg&level_level=$lev&alwaysRefire=false

23 = titan/script/2/Playbacks/FirePlaybackAtLevel?
    handle_location=PlaybackWindow_$arg&level_level=$lev&alwaysRefire=false
```

Finally: the examples already included in the config.ini should get you running quickly, and might even suffice for many situations. However, you are welcome to ask the author for a specific function, or – in turn – submit your strings to be included in the next version. Any feedback is highly appreciated!

[mapping]

Albeit looking most cryptic at a first glance, the entries in this section are built pretty much logical. They are constructed in a simple if – then – manner:

IF the software receives a particular MIDI message on channel x with note y
THEN it sends string no. m using n as argument (a possibly passed level value is not mentioned here)

More technically, the description is as follows:

- The MIDI message can be either CCh (Control Change), NOn (Note On) or NOf (Note Off). Technically there are two ways to send a Note Off command (either a direct Note Off, or a Note On with velocity = 0). Midityweb internally registers both as Note Off. Other MIDI messages (e.g. Prog Change) are currently not used by midityweb.exe.
- The MIDI channel is something to be set in the controller. Possible values are 1...16. If `omni_ch` is set (see section [general]) then the channel of incoming messages is ignored and all messages are mapped to the set omni channel.
- The MIDI note (NOf, NOn) resp. MIDI control (CCh) denotes which button you press on your controller. Possible values are 0...127.
- The three MIDI parts are separated with full stops, no spaces, e.g. `CCh.3.25`

If you are not sure which messages, channels and notes your controller is sending then see the next section.

- String no. is the number a particular string is given in the [strings] section (see above)
- Argument is an integer value which will be inserted as \$arg into the string. Only integer values are accepted. Each mapping entry is required to have a string no., a comma, and an arg value. If you don't need an arg value for a particular string then enter 0 as arg.
- The string/arg parts are separated with a comma and an optional space, e.g. `10, 1614`

Examples

`CCh.1.1 = 1, 1` MIDI Message ControlChange channel 1 control 1 calls string 1 with \$arg=1. With our default strings (see above) this would fire playback user no. 1. The velocity value is automatically mapped to \$lev.

`CCh.1.9 = 10, 1605` MIDI Message ControlChange channel 1 control 9 calls string 10 with \$arg=1605. As for the default strings this would set a master level, referring to the Titan ID 1605 which in new shows is the Grand Master (see below). Again the velocity is silently being passed as \$lev.

NO_n.1.21 = 2, 3

MIDI Message Note On channel 1 note 21 calls string 2 with \$arg=3. With the default strings this would swop playback userno. 3. Again the velocity is silently passed but as this string doesn't use \$lev it is not used here at all. Make sure to use ClearSwop as well when using swop (see next example).

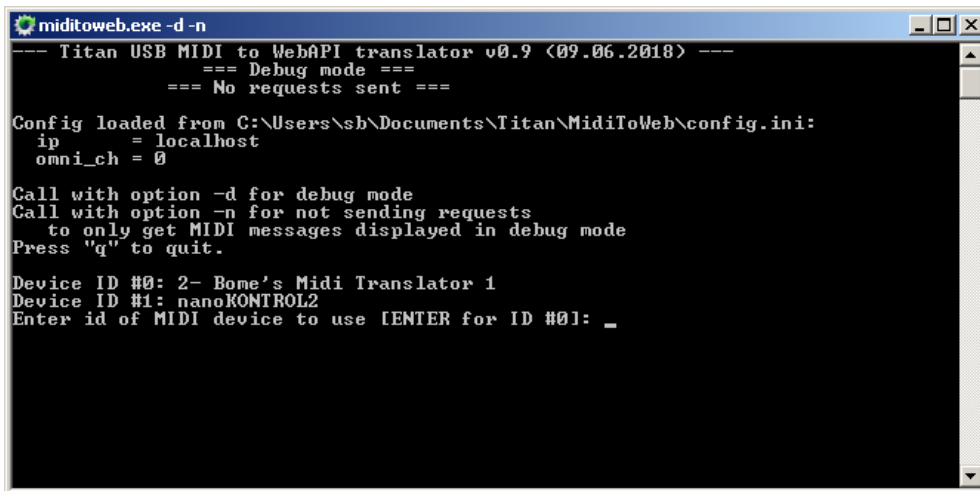
NO_f.1.21 = 3, 3

MIDI Message Note Off channel 1 note 21 calls string 3 with \$arg=3. With the default strings this would 'unswop' playback userno. 3. Again the velocity is silently passed but as this string doesn't use \$lev it is not used here at all.

Finding out MIDI messages

More often than not you will need to find out the very MIDI messages your controller is sending – maybe it is set to some weird settings which cannot be changed for other reasons, maybe the manual is missing, who knows. For such cases, miditoweb.exe offers a neat way to get the required data: the debug mode.

Simply start the program with the option `-d` (or both, `-d -n`, which also disables any http requests - this is useful when you want to prevent Titan from reacting to any unintended commands, or if you haven't Titan running and don't want to wait for answers which will never come.) To make your life easier, there is already a link MIDI to Web (Debug) in your program menu. Simply click it, and the program starts in Debug mode.



```
miditoweb.exe -d -n
--- Titan USB MIDI to WebAPI translator v0.9 (09.06.2018) ---
=== Debug mode ===
=== No requests sent ===

Config loaded from C:\Users\s\b\Documents\Titan\MidiToWeb\config.ini:
ip = localhost
omni_ch = 0

Call with option -d for debug mode
Call with option -n for not sending requests
to only get MIDI messages displayed in debug mode
Press "q" to quit.

Device ID #0: 2- Bome's Midi Translator 1
Device ID #1: nanoKONTROL2
Enter id of MIDI device to use [ENTER for ID #0]: _
```

This shows miditoweb started with `-d -n` options – we get the message `=== Debug mode ===` and `=== No requests sent ===` and are prompted for the device ID. Here, we have two MIDI devices: Bome's MIDI translator, ID #0, and a nanoKONTROL, ID #1. We use the nanoKONTROL, enter 1 and hit <ENTER>.

```
miditoweb.exe -d -n
--- Titan USB MIDI to WebAPI translator v0.9 (09.06.2018) ---
=== Debug mode ===
=== No requests sent ===

Config loaded from C:\Users\s\b\Documents\Titan\MidiToWeb\config.ini:
ip = localhost
omni_ch = 0

Call with option -d for debug mode
Call with option -n for not sending requests
to only get MIDI messages displayed in debug mode
Press "q" to quit.

Device ID #0: 2- Bome's Midi Translator 1
Device ID #1: nanoKONTROL2
Enter id of MIDI device to use [ENTER for ID #0]: 1
Using MIDI device #1

Debug MIDICallback:
dwInstance: 0
Handle: 0x77fd70
dwParam1 (Raw Data): 0
byteArray[0]: 0
byteArray[1]: 0
byteArray[2]: 0
byteArray[3]: 0
dwParam2 (timestamp): 0
uMsg:961
-----
Debug MidiReceiver():
The unsigned long, result, value was 0
MIM_OPEN: 961
MIM_CLOSE: 962
MIM_DATA:963
```

The software confirms that it is using device #1 and displays the first debug infos: `uMsg:961` tells us: the message is 'device interface is opened', hence ready to receive a message.

We now press a button on the controller...

```
Debug MIDICallback:
dwInstance: 0
Handle: 0x2e3fa0
dwParam1 (Raw Data): 8331440
byteArray[0]: 0
byteArray[1]: 127
byteArray[2]: 32
byteArray[3]: 176
dwParam2 (timestamp): 144578
uMsg:963
-----
Ch. 1 Control Change
Searching for CCh.1.32
Debug MIDICallback:
dwInstance: 0
Handle: 0x2e3fa0
dwParam1 (Raw Data): 8368
byteArray[0]: 0
byteArray[1]: 0
byteArray[2]: 32
byteArray[3]: 176
dwParam2 (timestamp): 147647
uMsg:963
-----
Ch. 1 Control Change
Searching for CCh.1.32
=
```

After having pressed and released a button, we see two sets of debug messages. Each one consists of

- "Debug MIDICallback" and the raw MIDI data
- The translated MIDI message
- The mapping string to be searched

The upper example reads

- MIDI data
 - o Control Change channel 1 (translated from byte 3: 176)
 - o Control (note) 32 (from byte 2: 32)

- Value/velocity 100% (from byte 1: 127)
- The translated message: Ch. 1 Control Change
- Searching for CCh.1.32 – this is the mapping the software tries to map – and as there is no entry for this MIDI message in the current config.ini, there is no string found.

The lower example is almost similar, only the value (byte 1) is different. Hence the mapping would be the same: CCh.1.32

Let's try another control – this time a fader:

```
Debug MIDICallback:
dwInstance: 0
Handle: 0x2e3fa0
dwParam1 (Raw Data): 2032048
byteArray[0]: 0
byteArray[1]: 31
byteArray[2]: 1
byteArray[3]: 176
dwParam2 (timestamp): 398073
uMsg:963
-----
Ch. 1 Control Change
Searching for CCh.1.1
HttpRequest:
titan/script/Playbacks/FirePlaybackAtLevel?userNumber=1&level=0.244094&bool=false
Debug MIDICallback:
dwInstance: 0
Handle: 0x2e3fa0
dwParam1 (Raw Data): 2097584
byteArray[0]: 0
byteArray[1]: 32
byteArray[2]: 1
byteArray[3]: 176
dwParam2 (timestamp): 398073
uMsg:963
-----
Ch. 1 Control Change
Searching for CCh.1.1
HttpRequest:
titan/script/Playbacks/FirePlaybackAtLevel?userNumber=1&level=0.251969&bool=false
```

By now this should look familiar:

- The MIDI data denote
 - Control change channel 1 (byte 3)
 - Control/note number 1 (byte 2)
 - Value something below and above 25% (byte 1 – in the first packet 31, in the second packet 32)
- Of course this is translated to 'Ch. 1 Control Change'
- The search string for mapping is now Ch.1.1
- And this time we got a hit (see examples above in the [mapping] section) – this is mapped to the string

```
1 = titan/script/Playbacks/FirePlaybackAtLevel?userNumber=$arg&level=$lev&bool=false
```

with \$arg being replaced with 1 (our user number) and \$lev being replaced with the normalized level (control value / 127). And if we hadn't started the program with -n then our playback #1 would have been set to approx. 25% ☺

Web API commands, Titan IDs and Locations

Web API command strings, Titan IDs (the internal numbering system) and Locations (references to specific items, e.g. fader playbacks or window buttons) are not covered here. The main source of information regarding these topics is the Titan Web API documentation at <http://api.avolites.com/11.0>. Also, there are some facebook groups: [Avolites Programmers and Users](#) and [Avolites Webapi](#). A designated forum is provided

at <http://forum.avolites.com/viewforum.php?f=21>. And the author runs a wiki to streamline all the available information at <http://www.avolites.de/wiki/> (in particular the Custom Macros section holds lots of infos).

However, while programming miditoweb the specific problem to find out Titan IDs and locations became apparent again, and while there are already a few ways to gather these data, neither is obvious or particular comfortable.

This is how gettitanids.htm got into the play. This is a very simple website which, being called at the same machine where your Avolites Titan Mobile software is running, retrieves and displays all your handles – with Legend, UserNumber, Titan ID and Location in the current show. For ease of use this is also linked in the Avolites program group – simply click the link, and (after some seconds) the browser should show something like this:

Titan IDs

This lists the assigned Titan IDs in the current show if Titan Web API is currently running on the local computer (obviously only available with Titan Mobile from v10 on.)

- 01.06.18 - first version published (v0.8)
- 02.06.18 - compose and display location string (v0.85)

These location strings have been tested for the Web API in v11 - they are different in v10.

Handles patched onto the Titan Mobile Wing are not reported via /titan/handles at all.

© 2018 Sebastian Beutel, s.beutel@avolites.de

Showname: "MidiTriggers"

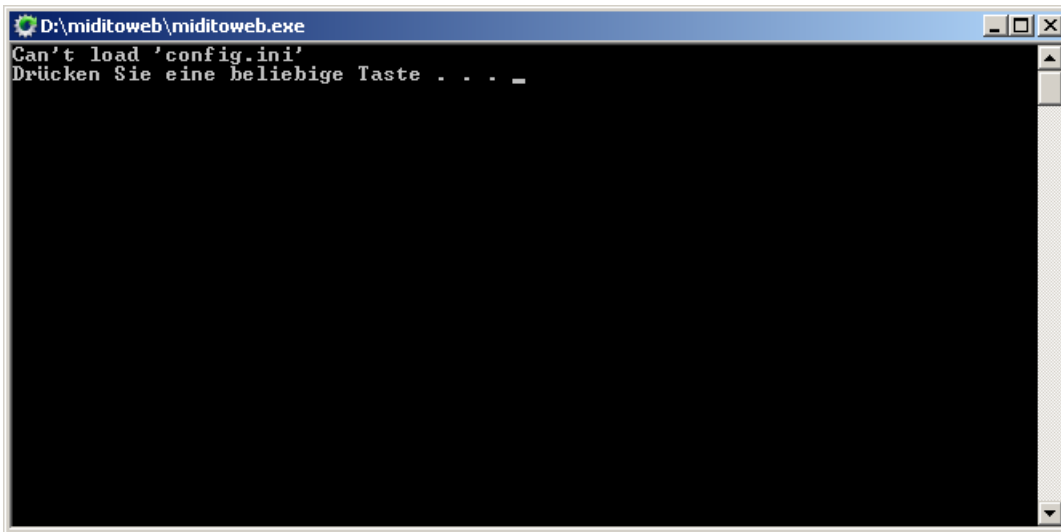
Number of handles: 19

Legend	Usernumber	Type	Titan ID	Location	Computed Web API Location string
Cue Playbacks[1]	1	cueHandle	1829	{"group":"RollerA","index":0,"page":0}	Playbacks_1_1
Cue Playbacks[2]	2	cueHandle	1831	{"group":"RollerA","index":1,"page":0}	Playbacks_1_2
Cue Playbacks[3]	3	cueHandle	1833	{"group":"RollerA","index":2,"page":0}	Playbacks_1_3
Cue Playbacks[4]	4	cueHandle	1835	{"group":"RollerA","index":3,"page":0}	Playbacks_1_4
Cue Playbacks[5]	5	cueHandle	1837	{"group":"RollerA","index":4,"page":0}	Playbacks_1_5
Cue Playbacks[6]	6	cueHandle	1839	{"group":"RollerA","index":5,"page":0}	Playbacks_1_6
Cue Playbacks[7]	7	cueHandle	1841	{"group":"RollerA","index":6,"page":0}	Playbacks_1_7
Cue Playbacks[8]	8	cueHandle	1843	{"group":"RollerA","index":7,"page":0}	Playbacks_1_8
Cue Playbacks[9]	9	cueHandle	1845	{"group":"RollerA","index":8,"page":0}	Playbacks_1_9
Cue Playbacks[10]	10	cueHandle	1847	{"group":"RollerA","index":9,"page":0}	Playbacks_1_10
Grand Master		masterHandle	1605	{"group":"RollerB","index":9,"page":0}	Playbacks_2_10
Rate Grand Master		rateMasterHandle	1609	{"group":"RollerB","index":5,"page":0}	Playbacks_2_6
BPM Master 1		rateMasterHandle	1612	{"group":"RollerB","index":6,"page":0}	Playbacks_2_7
All Dimmer	1	groupHandle	1823	{"group":"Groups","index":0,"page":0}	Groups_1_1
[4x Dimmers]	2	groupHandle	1824	{"group":"Groups","index":1,"page":0}	Groups_1_2
	1	fixtureHandle	1819	{"group":"Fixtures","index":0,"page":0}	Fixtures_1_1
	2	fixtureHandle	1820	{"group":"Fixtures","index":1,"page":0}	Fixtures_1_2
	3	fixtureHandle	1821	{"group":"Fixtures","index":2,"page":0}	Fixtures_1_3
	4	fixtureHandle	1822	{"group":"Fixtures","index":3,"page":0}	Fixtures_1_4

You should be able to use the Titan ID straight away. 'Location' is what Titan gives as location for this handle. As the WebAPI expects this in a serialized form (just one string), this is then converted in the Location string which can be used as handle_location in your strings.

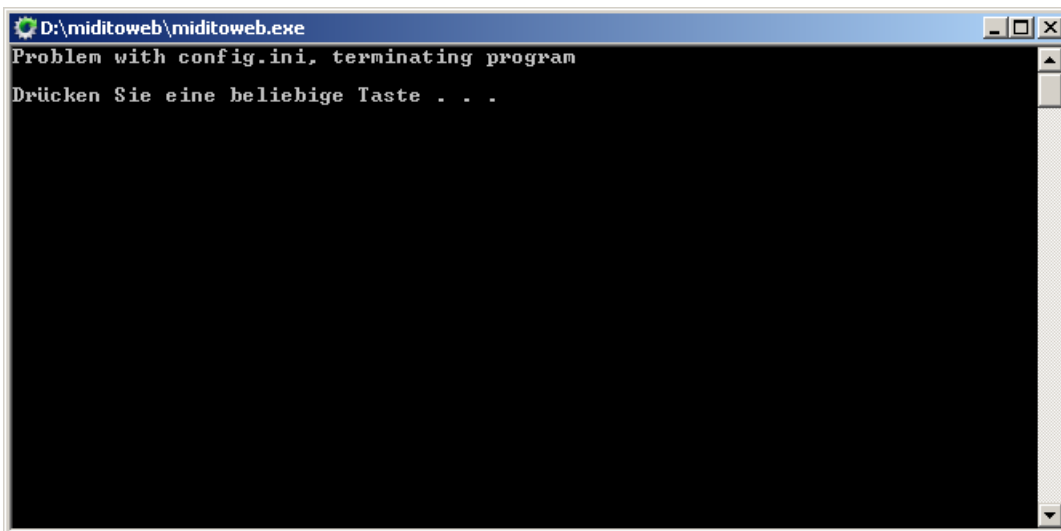
Possible errors

If the program doesn't find a config.ini, you get a message `Can't load 'config.ini'` and the program terminates itself. Put a config.ini in the correct place and try again:



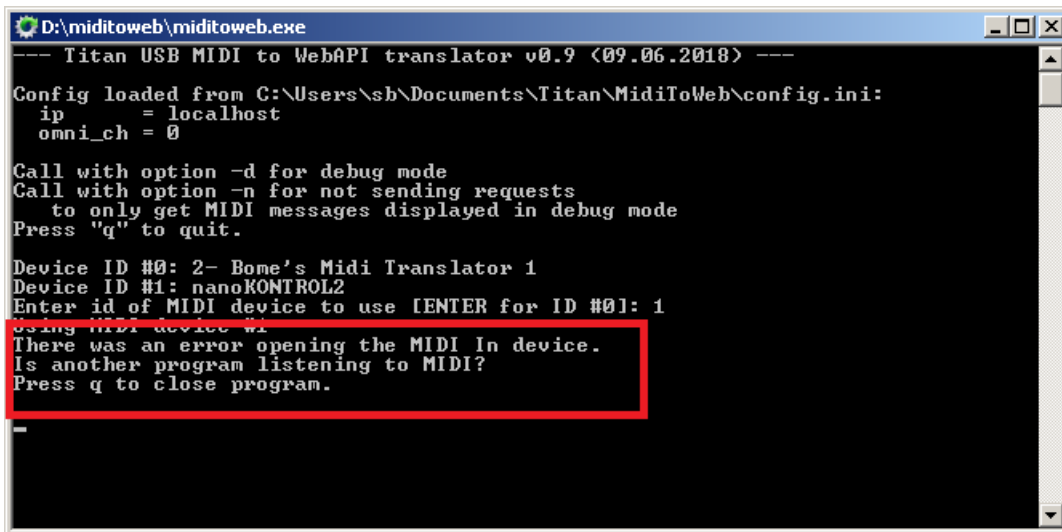
```
D:\miditoweb\miditoweb.exe
Can't load 'config.ini'
Drücken Sie eine beliebige Taste . . . _
```

If there is a config.ini but it doesn't contain correct data then you get the message `'Problem with config.ini, terminating program'`, and the program terminates itself. Fix the config.ini or install a new one and try again.



```
D:\miditoweb\miditoweb.exe
Problem with config.ini, terminating program
Drücken Sie eine beliebige Taste . . .
```

If there is a problem to open the specified MIDI device – wrong device ID entered, already in use by another program or another instance of miditoweb.exe – then the program throws a message **There was an error opening the MIDI In device! Is another program listening to MIDI?** Close the program (by pressing <q>), make sure only one program uses a MIDI in device at a time, and enter the correct device ID.



```
D:\miditoweb\miditoweb.exe
--- Titan USB MIDI to WebAPI translator v0.9 (09.06.2018) ---
Config loaded from C:\Users\s\Documents\Titan\MidiToWeb\config.ini:
ip = localhost
omni_ch = 0
Call with option -d for debug mode
Call with option -n for not sending requests
to only get MIDI messages displayed in debug mode
Press "q" to quit.
Device ID #0: 2- Bome's Midi Translator 1
Device ID #1: nanoKONTROL2
Enter id of MIDI device to use [ENTER for ID #0]: 1
Opening MIDI device #1
There was an error opening the MIDI In device.
Is another program listening to MIDI?
Press q to close program.
```

Differences between MidiToWeb and MIDI triggers

Of course MIDI triggers inside Titan would be the first choice when it comes to using MIDI with Titan. However, MidiToWeb might have its reasoning:

- It works with class-compliant devices while MIDI triggers only work with standard 5pin MIDI cable
- MIDI triggers are limited to programmed handles while MidiToWeb can use the whole range of commands the Web API offers. On the other hand, MIDI triggers have an easy-to-use learn function while MidiToWeb needs to be configured through editing the config.ini. Also, triggers may use the handle's key profile whereas MidiToWeb always deals with the programmed content itself.
- You can store the config.ini and copy it to other Titan installations while MIDI triggers (currently) run only with a specific console (they do not copy to other consoles)
- Using the WebAPI opens up the possibilities to using multiple devices, running the signal over large distances etc.

Limitations

MidiToWeb can be demanding on resources – the signal simply goes a longer way: from your MIDI device into the computer, then being 'translated', then sent to Titan WebAPI, to be finally processed by Titan. This takes its time – and depending from your computer's power you might restrict yourself to using push buttons and no faders.

Also, MidiToWeb requires a good knowledge of the WebAPI commands. There is already lots of information available, but this is not for the faint-hearted. In particular the idea of separated programmers can raise questions – in theory you can do most of the things by WebAPI which you usually do on the console – you simply do not see any result.

And currently I have no idea how to accomplish a few things if this is possible at all:

- Calling macros which are stored on the console by WebAPI
- Calling Go globally (like pressing the Go button) by WebAPI
- Set certain parameters (like the global palette fade time) by WebAPI
- Tap Tempo by WebAPI
- Select/Deselect fixtures and groups by WebAPI

I'll be happy to integrate solutions for these and other questions into MidiToWeb – just let me know if you found something which you'd love to see integrated.

Running multiple controllers

You can, however, run miditoweb in multiple instances. Simply make sure you assign another device ID per instance. All instances will use the same config.ini, the same mapping and strings, and will send their commands to the same Titan installation.

Finally

As stated in the LICENSE.txt this software comes with neither warranty nor liability – use it at your own risk. However, any feedback is highly appreciated, and the author is happy to implement new ideas if appropriate.

If you have something to contribute please write an email to Sebastian Beutel, s.beutel@avolites.de.

Thanks in advance for any feedback.

Happy Avolife ☺

© 2018 Sebastian Beutel