



ORGELPARK

Interface Manual

Documentation for interfacing with the digital console of the Sauer and Utopa organs.

Version #4
Wouter Snoei 2023

Index

Index	3
1 About this manual.....	4
2.1 Tools of choice	4
2.2 What is MIDI	5
2.3 What is OSC.....	5
3 How to control the organ.....	8
MIDI	8
OSC.....	9
Mixed	10
4.1 Setting up and using MIDI.....	11
4.2 Setting up OSC	11
4.3 Using OSC with Ableton Live / Max4Live device	12
4.4 Using MIDI/OSC with the Orgelpark MIDI to OSC application	13
4.4.1 Playing notes.....	14
4.4.2 Chord memory system	16
4.4.3 Controlling registers.....	17
4.4.4 Register presets	18
4.4.5 Controlling settings	19
4.5 Using OSC with other software	20
5 OSC implementation	23
5.1 Notes.....	23
5.2 Registers Utopa.....	24
5.3 Registers Sauer.....	25
5.4 Divisions Sauer	26
5.5 Layer settings	27
5.6 Utopa Tremolo	29
5.7 Swell pedals	30
5.8 Setzer	30
5.9 Single commands	31
6 Console MIDI implementation	32
6.1 Notes.....	32
6.2 Swell pedals	32
6.3 Setzer	32
6.4 "The rest"	33
7 List of registers and ranges	35

1 About this manual

This document will give directions on how to control the Utopa and Sauer at the Orgelpark via MIDI and OSC, to be able to play the instruments via computers and external (MIDI) controllers. The manual assumes general knowledge about the instruments themselves and the digital console. Distributed with this manual are files with data, graphs, patches, plugins and code examples for various platforms. Altogether I hope to give you, the reader, as many tools and information as I can for you to be able to get creative and control these magnificent instruments in ways no-one ever imagined before. Or simply to make music with them.

As the console itself is basically a computer, and the organ is always in development, things explained in this manual may at some point be caught up by reality, as new functions, improvements and changes may be applied to the instruments. I will try to keep the manual up-to-date, so therefore it is wise to check if you have the latest version of it.

2.1 Tools of choice

The digital organ console, as created by the kind people of Sinua and Engelbertse Orgelbouwers, was fitted with a **MIDI** interface, and an **OSC** network protocol. Both these interfaces can be used to control the organ, and each has their pros and cons. As a user and software engineer I would say the OSC protocol is most fitted to control all aspects of the instruments. Of course, MIDI is a more widely accepted and supported protocol, but unfortunately it is as a system not flexible enough to be able to support all the functionality of the instruments in this case. Basic functions such as playing notes on the manuals and pedals, and switching between settings on the console's Setzer can be done via MIDI, but as soon as we get to setting registers or changing layer properties it becomes harder if not impossible to do via MIDI. This is where OSC comes in, allowing full and direct control on all parameters, and even allowing the user to play notes in individual layers instead of whole manuals. It could happen, in some cases that both MIDI and OSC are used alongside each other. The choice of protocol also depends on the software that you intend to use. Here follows a list of known (and less known) software packages that should be able to control the organ via MIDI and/or OSC:

Software	MIDI	OSC
Ableton Live	Yes	Yes *)
Logic Pro	Yes	Via Max patch **)
Sibelius	Yes	Via Max patch **)
Other DAW/Notation software	Yes	Via Max patch **)
Max	Yes	Yes
SuperCollider	Yes	Yes
OpenMusic	Yes	Yes ***)
Cabbage / Csound	Yes	Yes
Qlab	Yes	Yes
JavaScript, Python etc.:	Yes	Yes

*) using the M4L instrument supplied with this manual

**) using the MIDI to OSC conversion Max patch / application supplied with this manual

***) untested

There are various iPad/iPhone/tablet apps on the market supporting OSC. However, the OSC protocol created for and used by the organ console is not easily created by any of these apps so far. For example, it should be possible to use TouchOSC (mk2) but it would require some work to create a working patch to control the organ.

2.2 What is MIDI

MIDI is short for Musical Instruments Digital Interface. It was invented in the 1980-ties as a means for combining multiple synthesizers, and later on became more popular applied in sequencers and music software as a way to control those hardware synthesizers. In the current situation MIDI is usually used the other way around; as a way to control computer software via hardware controllers. The MIDI protocol is quite simple in terms of computer science. It uses only 8-bit values at a fixed speed. Where it falls short is in the resolution of values and address space. The address space is also very much inspired on the architecture of keyboard-driven synthesizers, which is a limitation in the current field where much more complex and different kinds of instruments exist. Speed is also an issue; as soon as messages start becoming more complex (such as in System Exclusive messages or large chords) there can be perceivable delays. The main pro for using MIDI is in that same simplicity, which makes it a very reliable system. Basically, unless the cable is broken, MIDI always just works as soon as it is connected.

“Classic” MIDI comes in the form of a 3-pole cable with a DIN plug, connected to a dedicated MIDI interface. The information only travels one-way, so if back-and-forth connection is required two cables are needed. Many hardware MIDI controllers however have a built-in MIDI interface and connect directly via a single USB cable to a computer. There are also various network MIDI protocols, which allow MIDI information to travel over a standard (possibly WIFI) computer network. The MIDI connection of the digital console at Orgelpark uses the classic DIN MIDI connection, so in order to connect it to a computer a separate MIDI interface device is required. There is one available at the Orgelpark. More about MIDI can be found at <http://www.midi.org/>.

2.3 What is OSC

In recent years a reasonably stable standard for music communication came to be, named Open Sound Control. OSC is a network protocol with a standardized way of combining addresses, parameter names and values. The aim of the system is the same as with MIDI; communication between electronic music software and hardware. The approach is different in the sense that there is much more freedom in the way messages are composed. Messages in OSC usually include text, can have multiple values and can be almost any size. The only

thing OSC defines is the way these messages are technically transported between sources and receivers, and what data types they can be in. These types are: String (text), 32-bits Integers (whole numbers), 32-bits Floating Point numbers (fractional numbers with decimal point) and so-called “data-blobs” containing arbitrary sized amounts of data. Also specified in the OSC format are time tags, pointing to a specific moment in time where a received message should be evaluated, but not so many existing OSC-enabled applications are able to use those (they are also a bit inconvenient when using between multiple computers as they need to have their clocks precisely aligned for it to work).

While the data-types that can be sent are defined by the OSC standard, the exact composition of text and values to send and receive is up to individual manufacturers and software builders. This usually depends on what is to be controlled. By convention, messages tend to start with a text part in the style of a typical URL address, with forward slashes (“/”) in between words (similar to web addresses), stating which specific parameter or function the message should apply to. Then this is followed by one or more numbers (float / integer) or separate words, also depending on the type of parameter. A message for setting for example the frequency of an oscillator to 440Hz could look like this:

```
[ “/oscillator/freq”, 440 ]
```

But it could also look like this:

```
[ “/oscfreq”, “440Hz” ]
```

Or whatever the builder of the oscillator in question decided it should be. This means that to be able to control something via OSC one must first know what the messages for that specific instrument or device should look like. What exact words are to be used, and what types of values can be sent with them. Therefore, a set of OSC messages that works for one specific device or program, doesn’t necessarily work for another. Sinua, the builders of the Utopa/Sauer organ console, designed an OSC language to fit the specific needs and capabilities of this instrument. This manual will provide you with all the details of it in section [5 OSC implementation](#).

In the freedom and slightly anarchistic quality of the OSC protocol lies also the reason for the creators of the Orgelpark organ console to use it. Where MIDI falls short in simply not having enough types of messages to control all features of the organs, OSC can have a nearly unlimited amount of individual message types. Also, because messages can be of any size, it creates the possibility to make whole chords or clusters, or for example set all registers at once, with a single message. The OSC language that Sinua invented for this instrument is very flexible and at the same time quite simple and straight-forward to learn.

The connection required for OSC is only a network cable. If multiple computers are to be connected to the organ, an *ethernet switch* is required. The Orgelpark has a number of these available.

The OSC messages can also go over WiFi, but in the case of the Orgelpark you would need to bring your own WiFi router for that. Please note that WiFi is generally a less reliable way of

transporting OSC messages for the Orgelpark organ, as there is a chance of drop-outs, causing either missed or stuck notes, especially if many notes are played at the same time.

As OSC is a network protocol it shares the same pros and cons with other network-based solutions, in particular one we all know quite well; the internet. The speed of the OSC connection depends entirely on the network speed itself, and can be influenced by other programs and/or computers using the network. There is no guaranteed speed for a message to travel between sender and receiver, although in the case of a wired network with little traffic there shouldn't be any perceivable latency or difference in timing. Things get different at high traffic, and this traffic can also be caused by sending many OSC messages at the same time. At a certain point the network traffic will stall and you will get the typical stuttering that also happens with YouTube clips playing on slow WiFi. Also, there is no 100% guarantee that all your messages will arrive at the receiver. As said, these problems only occur in situations where there is much network traffic. It is usually advised to do OSC communication on a network that isn't connected to the internet, to make sure the only thing that travels through the network cables is actual OSC messages. And, if you intend to do really fast things or a send a lot of messages, beware of the fact that some may arrive later or not at all, and that timing in such cases may become less tight. That said, the speed of OSC messages in the case of the system at the Orgelpark exceeds that of MIDI by a large factor.

When dealing with OSC software, the user sometimes gets a choice between UDP and TCP modes. This is a technical detail, OSC can operate in both modes but the implementation of the Orgelpark console only supports UDP. Also, some OSC hosts support "bundled messages" and time stamps, but these are also not supported by the Orgelpark organ.

More info about OSC can be found at: <http://opensoundcontrol.org/>.

3 How to control the organ

To be able to play or control the organ via an external device or computer, first a choice needs to be made between OSC and MIDI. As explained above, it depends a bit on the software you are (comfortably) using, but also on your goal (which may actually influence your choice of software as well). The way things are at the current situation is as follows:

MIDI

The MIDI implementation of the organ is there somewhat by coincidence. The console itself, including the manuals, pedals and all the buttons and knobs, is in fact a large MIDI controller. Knowing what that device sends to the system is the same as knowing the MIDI implementation. So theoretically, everything you can do by hand with the console, can be done also via MIDI. But, it has to be done really in the same way. For playing the keyboards (manuals and pedals) this is no problem, as sending MIDI notes on the correct channels (1 to 4) will have the same effect as playing them by hand. If this is all you need, then MIDI can be the way to go. Section [6 Console MIDI implementation](#) describes how this is done.

If you want to do anything with registers however, things get more complex. Each of the register switches has its own MIDI note (a full list of the exact notes and messages to be sent for each switch can be found elsewhere in this manual). In case of the Sauer registers this is quite straightforward, as they are not dependent on the layer you are in. But in case of the (black) Utopa registers it is an issue, as the state of these switches changes depending on which layer is selected on the touch screen. Unfortunately, there is no way for an external computer or device of knowing which layer is currently selected, so flipping the switch via MIDI doesn't have a predictable effect; it may set that register in *some* layer, or do nothing at all when no layer is selected. A workaround for this could be to first send another MIDI message that presses the button for the desired layer on the desired manual, and then send the message for the register switch. But still, the actual behavior in that case would depend on the current state of the console, because if that layer happened to be selected *already*, then hitting the button via MIDI will actually cause it to deselect. Perhaps with more experimentation a solid way of controlling the Utopa registers via MIDI can be found, but I wouldn't really recommend using MIDI for this, as whatever outcome such experimentation may have, it certainly will be a complex and inflexible set of messages that needs to be sent; i.e. more work, less music. The same goes for layer parameters such as transposition and pulse, and even the tremolo knobs and switches. The only other parameters beside the notes on the manual that can be MIDI controlled consistently are the four continuous foot pedals. They have their own MIDI address and will behave consistently.

But, all is not lost, if you want to control the organ via MIDI and do need register changes, the Setzer is still at your disposal. Read how to control the Setzer via MIDI messages in the section [6.3 Setzer](#).

OSC

OSC control of the organ gets the most out of the instrument. Contrary to the MIDI implementation, the OSC implementation is not there by accident but deliberately designed for your controlling needs.

The easy route to “go OSC” is using **Ableton Live** and the Max4Live device that I’ve created for you (included with this manual). This will internally convert MIDI data in Live to the OSC language of the organ, and give you control over all possible parameters, all registers in all layers, and allow you to play individual layers instead of whole manuals. You can also couple external MIDI controllers to any parameter, and of course use Live’s built-in arpeggiators and other MIDI effects to your liking. Some tips and tricks about this plugin can be found further in this manual in section [4.3 Using OSC with Ableton Live / Max4Live device](#). Another thing good to mention is that OSC on the Utopa/Sauer organ currently is a one-way street; the user can send messages to the organ to control it, but the organ will never send anything back. It is not possible to get the current status of the organ, so it is also not possible for the Max4Live device to know which registers were already set before the plugin was opened. When using OSC in general it is advised to start from the (empty) initial setting of the organ.

The way to control the organ by OSC from an application that internally only supports MIDI (such as most DAW software), is to use our **MIDI to OSC application**. It comes with the manual and is available both as an app (macOS only) and a *Max patch* (cross-platform). The app can convert MIDI signals into OSC messages for the organ. It creates two virtual MIDI inputs on the machine you run it on, one for the notes and one for the settings and register controls (as said, a single MIDI connection doesn’t have enough flexibility to control all functionality of the organ). In your DAW, create an “external MIDI” channel, and choose one of the virtual MIDI inputs of the application to either play notes (optionally individually per layer) or send registrations and settings to the organ. The registrations are only possible with the app/patch, while sending MIDI notes can of course also be done via the regular MIDI to the organ. More information about the use of this app/patch can be found in section [4.4 Using MIDI/OSC with the Orgelpark MIDI to OSC application](#).

The (possibly) least easy but (maybe) more fun route is to dive into the OSC implementation yourself and use programming environments such as **Max** or **SuperCollider** to control your sound. Example patches in both languages are provided with this manual, including ready-to-go code to play Patterns in SuperCollider and the MIDI to OSC application/max patch mentioned above. This way of dealing with the organ could cost more work (depending on your own workflow and familiarity with said environments) but ultimately will allow you to get to the most unique ways of controlling and playing the instrument. Think in the lines of playing the instrument entirely via knobs and faders or sensors, use live sound input to control the behavior of sound, create generative and/or interactive compositions etc.. Also, if you want to get to the extremes of what the organ mechanisms can do, it may be required to program this yourself, as the OSC implementation has some hidden gems that allow wild performance (such as very tight clusters), and usually a programmatic approach allows you to fine-tune much more on performance-sensitive details.

Mixed

It is also possible to mix both protocols. For example; the organ console also has a MIDI output, which will send out the notes played on specific layers (this can be set in the layer properties on the console screen). This could be used for example by creating an empty layer (with no sounding registers) on one of the manuals, send the MIDI to Ableton Live, process it with MIDI effects (arpegiators etc.) and then via the Max4Live device send OSC back to other layers of the organ. This way the organ becomes its own MIDI controller. Also a combination of using OSC for registrations and settings and regular MIDI for playing notes is possible, and may be useful in situations where network traffic is high.

4.1 Setting up and using MIDI

To control the Utopa/Sauer console via MIDI from a computer, you will need at least:

- 1 MIDI cable
- 1 MIDI interface (usually an USB device, sometimes part of an Audio Interface)

The MIDI cable connects to the “out” of your MIDI interface, and to the “in” on the organ console. MIDI connections on the console are located next to the power cable, at the left-bottom side of the console. Depending on where your computer is standing you may need quite a long MIDI cable.

In your sequencer / DAW application: select the correct MIDI output for your MIDI interface. Now you can play **notes** on the first four MIDI channels:

MIDI Channel 1: Pedal section (P)

MIDI Channel 2: Manual 1 (lower manual / I)

MIDI Channel 3: Manual 2 (the middle manual / II)

MIDI Channel 4: Manual 3 (the upper manual / III)

The **Setzer** can be set using specific note messages. please refer to [section 6 Console MIDI Implementation](#), further in this manual.

To use the MIDI output of the console:

- connect a second MIDI cable to the “out” of the console and the “in” of your MIDI interface
- on the console, select a layer that you want to play and choose a midi channel for it in the on-screen settings tab

4.2 Setting up OSC

To set up OSC communication with the organ, a network cable (Ethernet Cat5e or better) is needed. The cable should be connected to your computer *) on one end and to the console on the other. The network connection on the console may be somewhat hard to find; it is a loose cable hanging under the console at the left side.

Once connected, the network will organize itself and give your computer an IP address. In order not to disturb this it is best to turn off the WiFi connection on your computer. The network settings on your computer should be set to “Use DHCP” (which in most cases they probably already are).

If multiple computers or devices are controlling the organ at the same time, an Ethernet / Network Switch (a device that basically acts as a “splitter” for network connections) has to be used. One of its ports should be connected to the organ via an Ethernet cable, and the other ports to the computers or devices that control the organ, each with their own Ethernet cable. The order of the ports has no influence on the functionality (so it doesn’t matter which device connects to which of the ports). There are



network switches available for this purpose at the Orgelpark with 8 ports, making it possible for a maximum of 7 computers/devices to be connected to the organ at the same time. For concert situations where different machines may control the organ per piece it is advisable to also use this switch so there doesn't need to be any reconnecting during the concert. Just make sure that the connected machines don't interfere with each other and don't send messages to the organ when another piece is playing.

If you are using OSC via the Max4Live device in Ableton Live or the MIDI to OSC application/Max patch, you are now ready to go. If you are using OSC in your own application you should know the OSC address of the console. The address is:

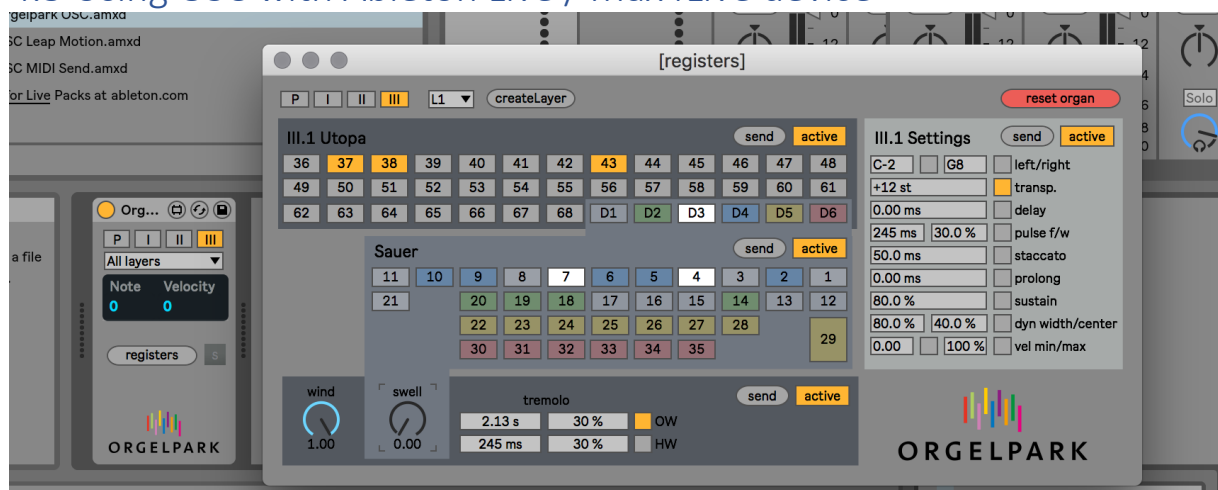
IP: 192.168.0.1

Port: 1803

Protocol: UDP

*) newer Apple MacBooks don't have built-in Ethernet connectors. To use OSC with such a machine you will need an Ethernet-to-USB-C or Ethernet-to-Thunderbolt adapter (or another device that allows Ethernet output from your machine).

4.3 Using OSC with Ableton Live / Max4Live device



If you are using Ableton Live and the Max4Live device that comes with this manual, this is how to proceed:

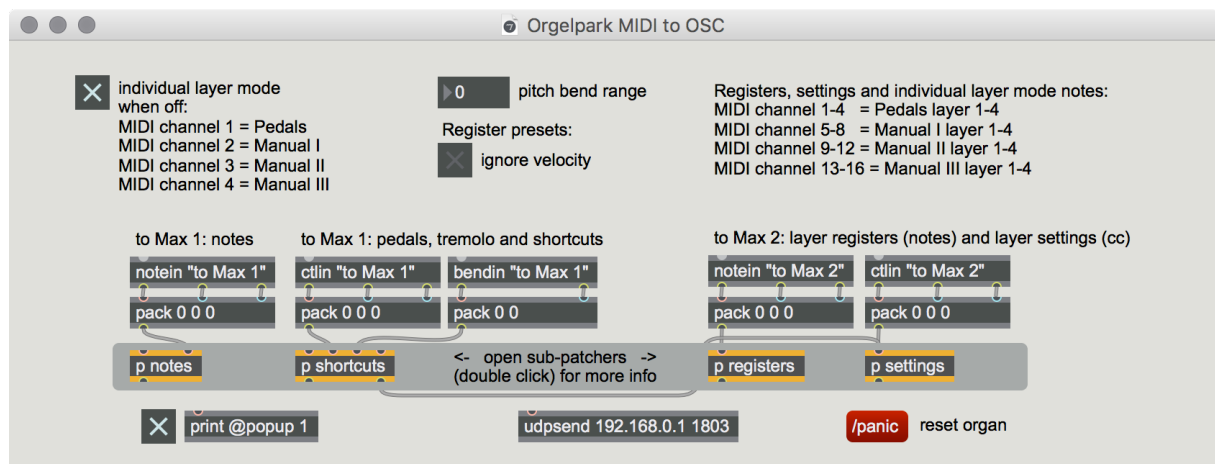
- open Ableton Live
- install the device on your system (if you haven't already done so)
- on a MIDI track, insert the device named "Orgelpark OSC" (which should be located in Max for Live -> Max MIDI Effect).
- Set the "MIDI To" field on the track to "No Output" (unless you want to output the same MIDI also to another instrument)

On the device you can set which manual and layer you want this particular MIDI track to play on. You can also set wind and pedal, and if you click the "registers" button you get access to all registers and layer parameters, per layer. Each of the buttons and knobs can be automated by recording their movement in Arrangement view, or adding envelopes to your

clip in Session view. Also, MIDI or Key mappings can be assigned. If you want to process your MIDI before going to the organ, simply add a MIDI effect before the device on the track.

When making register settings, be sure to enable the “active” button on in the registers window as well. Changes will be sent to the organ instantly, but you can also send all current settings of the selected layer at once with the “send” button. If you close your Set and open it again at a later moment, the device will automatically send the register settings for all layers and settings that have “active” on to the organ, so that it gets into the correct state (it is advisable to reset the organ before doing this). All settings will be saved only in your Set, and not on the organ console. If you use multiple instances of the M4L device in your Set, make sure that you don’t have overlapping register settings on the same layers in those devices. Good practice would be to only set the layer(s) that the device is playing on to “active”, so that that instance of the device controls and stores only those registers.

4.4 Using MIDI/OSC with the Orgelpark MIDI to OSC application



If you are working with a DAW or other application that only sends MIDI, but still want to make use of the possibilities of the OSC interface, you can use our MIDI to OSC application (*) or Max patch (supplied with this manual). It allows users to convert MIDI messages (notes, pitch-bend and controller data) to OSC messages for the organ.

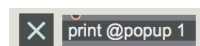
In short, the app/patch supports the following features:

- Note playback directly on individual layers, effectively creating a 16-part multi-timbral device
- Chord memory; grouping notes under individual keys
- Direct control over registers per layer
- Direct control over settings per layer
- Register presets, created and applied via MIDI (no storage on organ console needed)

The features mentioned above will be discussed in more detail on the next pages of this manual.

The application has two MIDI inputs; the first one for notes the second one for registers and settings. If you are using the application these are named “to Orgelpark MIDI to OSC 1” and “to Orgelpark MIDI to OSC 2”, and if using the Max patch it is “to Max 1” and “to Max 2”.

The application has a second window; the Max Console (and when using the Max patch this console is part of Max). When using the chord memory or register presets, messages can be seen in the Max Console informing about the status of your actions. It is also possible to show the messages sent to the organ by enabling the [x] button next to the [print @popup 1] box. This can be useful when working at home without the actual organ present.



Please note that the MIDI implementation of this application is *different* from that of the console itself. When controlling the console itself *directly* via MIDI please refer to [section 4.1 Setting up and using MIDI](#) and [section 6 Console MIDI Implementation](#).

The Max patch was created as an example of how OSC control can be implemented in Max. Please feel free to use (parts of) it in your own Max patches, or extend its capabilities.

*) the MIDI to OSC application is a standalone application for macOS, created with Max. On modern macOS machines however, the application will be blocked by the system’s protection services (macOS 10.14 and above). There are ways around this, but perhaps the best way is not to use the standalone application but the Max patch instead, and download and install Max to run it. If not licensed, Max can still run existing patches like this one, so it is not needed to buy the software. <https://cycling74.com/downloads>

4.4.1 Playing notes

The **notes** to play are to be sent to the first input of the program; “to Orgelpark MIDI to OSC 1” or “to Max 1”. The program has an *Individual layer mode* (on by default) for the **MIDI notes**. In this mode the MIDI are played directly on the layers instead of the manuals. It assumes that there are only 4 layers per manual, and they can be chosen via the MIDI channel:

MIDI Channels 1-4: Layer 1-4 of the pedal section (P)
MIDI Channels 5-8: Layer 1-4 of manual 1 (the lower manual / I)
MIDI Channels 9-12: Layer 1-4 of manual 2 (the middle manual / II)
MIDI Channels 13-16: Layer 1-4 of manual 3 (the upper manual / III)

If the “Individual layer mode” is off, the notes will play on all layers. In that case the channel order is as follows:

MIDI Channel 1: Pedal section (P)
MIDI Channel 2: Manual 1 (lower manual / I)
MIDI Channel 3: Manual 2 (the middle manual / II)
MIDI Channel 4: Manual 3 (the upper manual / III)

Also on the first input (“to Orgelpark MIDI to OSC 1” or “to Max 1”) there are some **global settings** as **MIDI Control** messages. These can be sent on any MIDI channel:

MIDI Controller (cc) 1: Tremolo OW; frequency
MIDI Controller (cc) 2: Tremolo OW; width / active *
MIDI Controller (cc) 3: Tremolo HW; frequency
MIDI Controller (cc) 4: Tremolo HW; width / active *
MIDI Controller (cc) 5: Swell pedal 1 (Wind motor Utopa)
MIDI Controller (cc) 6: Swell pedal 2 (Crescendo)
MIDI Controller (cc) 7: Swell pedal 3 (Swell box Sauer)
MIDI Controller (cc) 8: Swell pedal 4 (so far unassigned)

The **shortcuts** are connections to some settings that are normally controlled via the second input (“to Orgelpark MIDI to OSC 2” or “to Max 2”), but are convenient to have together with the notes:

MIDI Controller (cc) 64: Sustain
Pitch bend: Transpose **

*) tremolo is off at values 0 and 127, and on at values 1-126

**) the pitch transposition range can be set in the application/patch. Beware that this overrides the current transposition setting in the selected layers.

4.4.2 Chord memory system

The patch has a built-in system for assigning groups of notes to single note numbers. This enables users to use the special ability of the OSC implementation to send multiple notes in a single message. Doing so will make the organ respond faster and play tighter chords. The term chord in this case means a group of notes, which can have any size (i.e. a cluster of notes is also a chord, and this feature is especially handy for playing clusters faster and tighter than could be done with regular MIDI).

MIDI Note 127 (G8) on “to Orgelpark MIDI to OSC 1” or “to Max 1” is used for assigning notes. When this note is pressed down (note-on) the system will start recording notes to put in the chord. This means that any note that is played while note 127 is on will not be sounding but only collected for the chord to store. Then, when all notes you want to be in the chord are collected, release note 127 (note-off). The first next note that is played (apart from note 127) is where the chord will be stored.

Example; creating a chord note (sequence of note messages, velocity/channel are not used):

Type	Note	Max Console window shows:
noteOn	127	chord • recording
noteOn	72	chord • added note 72
noteOn	76	chord • added note 76
noteOn	79	chord • added note 79
noteOff	127	chord • waiting for note to assign
noteOn	60	chord • assigned to note 60

After this example is played note 60 (C3) will play a major triad (C4, E4, G4). This happens system-wide, on any channel/layer. The notes of the chord will be played with the velocity and channel of the assigned note.

To remove a chord from a note, simply follow the same procedure, but without playing any notes while note 127 is on.

Example; removing the chord note:

Type	Note	Max Console window shows:
noteOn	127	chord • recording
noteOff	127	chord • waiting for note to assign
noteOn	60	chord • assigned to note 60

This will remove the existing chord from note 60, and from there on playing note 60 will sound C3 again.

If you want to use chords in your MIDI score, the sequences to create them will have to be at the start of the score, before anything is playing. The Max patch / application in it's current version will forget the chords as soon as it is closed, so you need to create them again at your next session.

4.4.3 Controlling registers

Registers and settings can be controlled via the second MIDI input of the program; “to Orgelpark MIDI to OSC 2” or “to Max 2”. On this input the MIDI channel is used to assign a specific layer, in the same way as MIDI notes in “Individual layer mode”:

MIDI Channels 1-4: Layer 1-4 of the pedal section (P)

MIDI Channels 5-8: Layer 1-4 of manual 1 (the lower manual / I)

MIDI Channels 9-12: Layer 1-4 of manual 2 (the middle manual / II)

MIDI Channels 13-16: Layer 1-4 of manual 3 (the upper manual / III)

Registers are set via **MIDI Note** messages. The note number corresponds with the number of the register as found on the organ. Please refer to the [list of registers](#) in this manual or the separately supplied csv and pdf files to look up the names and numbers of the registers. A note with a **velocity** larger than 63 will turn the register on, and a note with lower velocity will turn it off. There are also extra notes for resetting registers and turning on and off sections in the layers. Note-off or velocity 0 notes will be ignored (except for note 0).

MIDI Note 0: reset / collect Utopa registers on layer *)

MIDI Notes 1-35: Sauer registers (global/any MIDI channel), velocity > 63 on, velocity < 63 off

MIDI Notes 36-68: Utopa registers on layer, velocity > 63 on, velocity < 63 off

MIDI Note 70: all Sauer divisions off on layer

MIDI Note 71-76: Sauer division 1-6 on layer, velocity > 63 on, velocity < 63 off

MIDI Note 80: reset all Sauer registers (global/any MIDI channel)

MIDI Note 81-86: reset Sauer registers in division 1-6 (global/any MIDI channel)

*) The “reset / collect” function of MIDI Note 0 is a special case. It can be used for resetting all registers on the layer, but also to set a number of registers via one OSC message. It is advised to do so when you want to set more than 3 or 4 registers at the same moment. The system works as follows:

When a note on **MIDI Note 0** is started (note-on) the system goes in to “collect” mode for that MIDI Channel/layer. All Utopa register notes (36-68) on that MIDI Channel/layer played after this will not set the register immediately, but will be collected in a list. Once MIDI Note 0 is released (note-off) this list will be sent to the organ, and all registers that are not in it will be turned off. If no registers are in the list then all registers are turned off.

Example (sequence of note messages):

Type	note	velocity	channel
noteOn	0	127	13
noteOn	38	127	13
noteOn	39	127	13
noteOff	0	(0)	13

Clears current Utopa register setting on Manual III, Layer 1 and enables registers 38 and 39 (Rohrflott 8', Quintathen 8')

4.4.4 Register presets

Similar to the function of storing chords (discussed in 4.4.2 Chord Memory System) there is a memory for register settings. **MIDI Notes 87 to 126** (Eb5 to F#8, on “to Orgelpark MIDI to OSC 2” or “to Max 2”) are reserved for this. To store a preset, first play **MIDI Note 127** (G8), then play the register notes you want to be in the preset (they can be Sauer registers (1-35), Utopa registers (36-68) or Sauer divisions (71-76)), with a velocity higher than 63, on any MIDI channel. After that, play the note to which you want to assign the preset (87 to 126). From then on, the preset is sent to the organ whenever you play that note, on the layer corresponding to the channel the note is played on.

Beware that also the **velocity** of the note is recorded, and the preset will only be sent when the *exact* same velocity is used. This means that you could (theoretically) store 126 different presets on a single note (each on a different velocity value), giving you a total of $40 * 126 = 5040$ (!) slots to store presets in.

Example; storing a register preset (any channel, note offs are ignored):

Type	note	velocity	Max Console shows
noteOn	127	<any>	register preset • recording
noteOn	38	127	register preset • added register 38 (Utopa)
noteOn	39	127	register preset • added register 39 (Utopa)
noteOn	90	1	register preset • Utopa 38 39
			register preset • assigned to note 90 velocity 1

From then on, playing note 90 with velocity 1 will set Utopa registers 38 and 39 (Rohrflott 8', Quintathen 8'), on the layer corresponding with the MIDI channel (see 4.4.3 Controlling Registers). The preset will turn off all other Utopa registers on that layer.

Note that if only Utopa registers are used in the preset (like in the example above), the Sauer registers and division settings remain untouched when using the preset. If only Sauer registers are used, the Utopa registers will remain untouched, and if only Sauer divisions are used, the rest will remain untouched. Registers and divisions of both organs can also be combined in a single preset.

In the current version, the presets remain in the memory of the Max Patch / application **until it is stopped**. The next time you open the patch you need to send the MIDI notes that create your preset(s) again to be able to use them.

Using the register preset system is advised when you have multiple large register changes in your piece. A register preset is sent as a single OSC message to the organ, and thus faster and more reliable than sending individual register on/off notes.

4.4.5 Controlling settings

Settings are controlled via **MIDI Control** messages (cc) on the second input (“to Orgelpark MIDI to OSC 2” or “to Max 2”). The numbering of these messages starts at 15 and the order is the same as that of the physical knobs on the organ console. Most of the settings on the console also have an on/off switch. In the MIDI to OSC application this is done automatically for convenience (i.e. to keep the control structure as simple and small as possible).

- MIDI Controller 15: Left MIDI note limit (value 0: off, value 1-127: note 1-127)
- MIDI Controller 16: Right MIDI note limit (value 0-126: note 0-126, value 127: off)
- MIDI Controller 17: Transpose (value 28-63: -36 to -1, value 64: 0/off, value 65-100: 1 to 36)
- MIDI Controller 18: Delay (value 0-127: 0ms (off) to 4096ms)
- MIDI Controller 19: Pulse frequency (value 1-127: 5ms to 5bpm, value 0: off)
- MIDI Controller 20: Pulse width (value 1-127: 0% - 100%)
- MIDI Controller 21: Staccato (value 1-127: 2ms to 1000ms, value 0: off)
- MIDI Controller 22: Prolong (value 0-127: 0ms (off) to 2000ms)
- MIDI Controller 23: Sustain (value 0-127: 0% (off) to 100%)
- MIDI Controller 24: Dynamic width (value 0-127: 0% (off) to 100%)
- MIDI Controller 25: Dynamic center (value 0-127: 0% to 100%)
- MIDI Controller 26: Minimum velocity (value 0-127: 0% (off) to 100%)
- MIDI Controller 27: Maximum velocity (value 0-127: 0% to 100%(off))
- MIDI Controller 28: MIDI output for layer (value 0: off, value 1-16: channel 1-16)

4.5 Using OSC with other software

With applications such as SuperCollider and Max it is possible to control the digital organ console directly via OSC. The OSC implementation (further in this manual) describes precisely how OSC messages sent to the organ should be composed, and what functionality can be accessed with them. OSC messages are (small) network packets in a format specified by the Open Sound Control 1.0 Specification (http://opensoundcontrol.org/spec-1_0). They typically consist of an address (text) and one or more values (numbers and/or text). The messages are sent to a network port on the organ console, which is specified using an IP address and a port number. As shown in section 4.2 Setting up OSC these are:

IP: 192.168.0.1
Port: 1803

Each application supporting OSC has a different way of specifying port and address, and also different ways of sending messages to them. In **SuperCollider**, for example, the address and port are set in a `NetAddr` object. Messages can be sent to the console via the `sendMsg` method and equivalent (look up the helpfile for `NetAddr` to see all possibilities). An example for starting a note 69 (A3) on the 3rd manual with velocity 80% could look like this:

```
n = NetAddr( "192.168.0.1", 1803 ); // define address and port
n.sendMsg( "/M3/V", 0.8, 69 ); // send note
```

In **Max** messages can be sent using the `udpsend` object. Messages can be sent by sending a list to the object. The above example could look like this:



In **Csound** an instrument that plays the above example could look like this:

```
instr 1
  OSCsend 1, "192.168.0.1", 1803, "/M3/V", "fi", 0.8, 69
endin
```

In **Pd** the above example could look like this:

```
0.8 69 [- click to send note
oscformat M3 V
list prepend send
list trim
connect 192.168.0.1 1803
disconnect
netsend -u -b
```

In **Processing**, using the **oscP5** library by Andreas Schlegel the example could look like this:

```
import oscP5.*;
import netP5.*;

OscP5 oscP5;
NetAddress myRemoteLocation;

oscP5 = new OscP5(this, 12000); //listening
myRemoteLocation = new NetAddress("192.168.0.1", 1803); // speak to

OscMessage newMessage = new OscMessage("/M3/V");
newMessage.add( 0.8 );
newMessage.add( 69 );
oscP5.send(newMessage, myRemoteLocation);
```

In **Python**, using the **pyOSC** module by Daniel Holth and Clinton McChesney (<https://github.com/ptone/pyosc>) the example could look like this:

```
#!/usr/bin/env python3
from OSC import OSCClient, OSCMessage

client = OSCClient()
client.connect( ('192.168.0.1', 1803) )

client.send( OSCMessage("/M3/V", [ 0.8, 69 ] ) )
```

In **JavaScript / Node.js**, using in this case the **node-osc** library from Myles Borins (<https://github.com/MylesBorins/node-osc>) the example could look like this:

```
var osc = require('node-osc');

var client = new osc.Client('192.168.0.1', 1803);

client.send('/M3/V', 0.8, 69, function () {
  client.kill();
});
```

When using OSC it is important to be aware about the difference in value types. The OSC implementation of the digital organ console uses three types: **String**, **Float** and **Integer**. In the case of the above example the message to be sent consists of a String, followed by one Float and one Integer.

Strings are usually defined in programming languages by the enclosing " " signs. As OSC doesn't support Symbols, in most languages it is ok to use Symbols instead of Strings, or to mix them (this also happens in the Max example above).

The distinction between Integers (whole numbers) and Floats is important, as they are interpreted differently by the console. For example, in the note message examples above, the 0.8 for velocity is a Float, and the 69 for note is an Integer. Most software automatically switches to Float as soon as a "." is notated in the number (i.e. 0.0 will become Float 0.0, and 0 will become Integer 0), but in some cases you will need to specify this manually (i.e. Csound).

Also, many OSC implementations regard the first String of the OSC message as "address". As shown in the various examples some programs treat this as a separate part, and others simply include it in the list.

5 OSC implementation

The OSC implementation describes how OSC messages sent to the digital organ console should be formatted. How this information can be used in various programming environments can be read in section [4.5 Using OSC with other software](#). For the implementation I'm using a standard notation derived from SuperCollider. OSC messages are Arrays containing Strings and values (Float / Integer).

5.1 Notes

The standard OSC message to start and end notes consists of at least three parts:

- An address String (manual, layer and note command)
- A velocity value (Float 0.0-1.0)
- One or more note numbers (Integer 0-127)

[address String, velocity value, ... note number(s)]

The **address String** exists in two variants:

`"/M<manual>/V"`

<manual>: manual number 0,1,2,3, where 0 means the Pedal section

`"/M<manual>/L<layer>/V"`

<manual>: manual number 0,1,2,3, where 0 means the Pedal section

<layer>: layer number 1 to 4 or more (depending on number of existing layers)

In the (latter) case where the layer is included in the address String the note will *only* play on the specified layer.

The **velocity value** is a Floating point number (0.0 - 1.0) which defines the strength of the note. Depending on this the organ valves will open slower or faster and layers that have velocity-switching enabled will react accordingly. A velocity value of 0.0 will end the note.

The **note number(s)** are Integers (0-127) with a note number in MIDI style (69 = A3). There can be any number of these in the message, and all of them will be either started or stopped (depending on the velocity value).

Examples:

`["/M3/V", 0.8, 69]` : starts note A3 with velocity 80% on Manual III (all Layers)

`["/M3/V", 0.0, 69]` : ends the above note

`["/M1/L1/V", 0.75, 60, 63, 67]` : starts three notes with velocity 75% on Manual I, Layer 1

5.2 Registers Utopa

The registers of the **Utopa** organ can be set per Layer (be aware that the Sauer registers are set with a different message type). The message to do so consists of at least two parts:

- An address String (manual, layer and register command)
- One or more register numbers (Integer)

[*address String*, ... *register number(s)*]

The **address String** points to the manual and layer where the registers should be set:

"/M<manual>/L<layer>/S"

<manual>: manual number 0,1,2,3, where 0 means the Pedal section

<layer>: layer number 1 to 4 or more (depending on number of existing layers)

The **register number(s)** are the numbers of the registers to be turned on or off. The number correspond to the numbers shown on the console, ranging from 36 to 68. When a register needs to be set, it's number should be included in the list. If a register needs to be closed, a negative number must be added. To close all registers (on the layer) at once a number 0 can be used:

36 to 38: turn on register 36 to 68

-36 to -68: turn off register 36 to 68

0: turn off all registers

It is common practice to send single positive / negative numbers in a message when only a small change is made. To replace the registration of a layer completely in one message, the message could start with a 0 and then be followed by the register numbers that should be (or stay) on.

Examples:

["/M3/L1/S", 37] : turn on register 37 (Principal 8') on Manual III, Layer 1

["/M3/L1/S", -37] : turn off register 37 (Principal 8') on Manual III, Layer 1







["/M2/L2/S", 0] : turn off all registers on Manual II, Layer 2

["/M1/L1/S", 0, 37, 38, 50, 61] : set registers 37, 38, 50, 61 on and the rest off on M. I, L. 1

Warning: Please note that in the current version of the Sinua organ console software the register setting via OSC can cause the console to **crash**. This happens when too many register change messages are sent in a short time (1000 or more per second), or if you send some settings immediately after a "/panic" message. If you want to create the effect of quick register changes it is better to use layers and notes for that.

5.3 Registers Sauer

The registers for the **Sauer** organ use two different message types. One is for turning the **individual registers** on and off, and the other is for assigning the divisions to specific layers. In order to make these you must know the 6 different divisions in which the registers are divided, and for each register in which division it is. The divisions are shown using colors on the organ console. They are as follows:

	Division 1 (grey):	12, 13, 15, 16, 17
	Division 2 (green):	14, 18, 19, 20
	Division 3 (white):	1, 3, 4, 7, 8, 11, 21 (motor ab)
	Division 4 (blue):	2, 5, 6, 9, 10, 19
	Division 5 (yellow):	22, 23, 24, 25, 26, 27, 28, 29 (tremolo)
	Division 6 (pink):	30, 31, 32, 33, 34, 35

The message for switching individual registers on and off consists of at least two parts:

- An address String (division and register command)
- One or more register numbers (Integer)

[address String, ... register numbers]

The **address String** points to the division in which the registers should be set:

`"/D<division>/S"`

<division>: division number (1-6)

The **register number(s)** are the numbers of the registers to be turned on or off. The number correspond to the numbers shown on the console, ranging from 1 to 35. Beware that the command only works if the stop you set with it is in the division chosen in the address String (i.e. setting register 14 can only be done on division 2). When a register needs to be set, it's number should be included in the list. If a register needs to be closed, a negative number must be added. To close all registers (on the division) at once a number 0 can be used:

1 to 35: turn on register 1 to 35 (depending on division)

-1 to -35: turn off register 1 to 35 (depending on division)

0: turn off all registers on division

Examples:

`["/D1/S", 15]` : turn on register 15 (Cello 8')

`["/D1/S", -15]` : turn off register 15

`["/D2/S", 0]` : turn off all registers in division 2 (i.e. 14, 18, 19 and 20)

`["/D5/S", 0, 23, 24]` : turn on only registers 23 and 24 in division 5.

5.4 Divisions Sauer

The OSC message for assigning a **division** of **Sauer** registers to a layer is similar to that of assigning Utopa registers. These divisions correspond with the colored square buttons in the middle of the console, in between the (black) Utopa register switches. The message consists of at least two parts:

- An address String (manual, layer and division command)
- One or more division numbers (Integer)

[*address String, ... division number(s)*]

The **address String** points to the manual and layer where the registers should be set:

"/M<manual>/L<layer>/D"

<manual>: manual number 0,1,2,3, where 0 means the Pedal section

<layer>: layer number 1 to 4 or more (depending on number of existing layers)

The **division number(s)** are the numbers of the divisions to be turned on or off (1-6, see description in 5.3 Registers Sauer). When a division needs to be assigned to a layer, it's number should be included in the list. If a division needs to be turned off, a negative number must be added. To close all divisions (on the layer) at once a number 0 can be used:

1 to 6: turn on division 1 to 6

-1 to -6: turn off division 1 to 6

0: turn off all divisions

Examples:

["/M3/L1/D", 3] : turn on division 3 (white) on Manual III, Layer 1

["/M3/L1/D", -3] : turn off division 3 (white) on Manual III, Layer 1

["/M0/L2/D", 0] : turn off all divisions on the Pedal section, Layer 2

["/M2/L1/D", 0, 2, 3, 4] : turn on only divisions 2,3,4 on Manual II, Layer 1

Example for setting a specific Sauer register on a specific layer:

["/D5/S", 0, 23] : turns on only register 23, followed by:

["/M3/L1/D", 5] : turns on the corresponding division on Manual III, Layer 1

Warning: Please note that in the current version of the Sinua organ console software the register setting, also for Sauer registers and divisions, via OSC can cause the console to **crash**. This happens when too many register change messages are sent in a short time (1000 or more per second), or if you send some settings immediately after a "/panic" message. If you want to create the effect of quick register changes it is better to use layers and notes for that.

5.5 Layer settings

It is also possible to control Layer parameters via OSC. Each of the parameters has their own command, and particular type(s) of values that can set it. Please note that the OSC implementation for layer settings is a bit buggy and sometimes inconsistent in the current version of the console, in particular the Sustain parameter which is not really usable.

In general the **layer setting** message consists of 2 or 3 parts:

- An address String (manual, layer and parameter command)
- A Float or Integer to set the value (sometimes both options are available)
- Optional: a String stating "on" or "off"

[address String, value (Float/Integer), "on" / "off" (optional)]

The **address String** points to the manual and layer, and which parameter should be set:

`"/M<manual>/L<layer>/<parameter>"`

`<manual>`: manual number 0,1,2,3, where 0 means the Pedal section

`<layer>`: layer number 1 to 4 or more (depending on number of existing layers)

`<parameter>`: name of the parameter to be set

The **value** can be a Float or Integer, range depending on the parameter that is being set.

The **"on" / "off"** String can be added to turn on and off the parameter. This applies to every parameter, but there are some cases where it renders unexpected results or is not necessary (noted below). Note that while it is possible to add "on" / "off" to a message with a value, it is *not* possible to leave out the value and send only the "on" / "off".

The following parameters can be set using the exact names - case sensitive - as parameter command (in order of their appearance on the organ console):

Left (left note limit)

value: Integer (0-127) sets the note limit in MIDI note format (69 = A3)

"on" / "off": enables / disables the limit

Right (right note limit)

value: Integer (0-127) sets the note limit in MIDI note format (69 = A3)

"on" / "off": enables / disables the limit

Transpose

value:

- Integer (-36 to 36) sets the transposition amount in semitones
- Float (-1.0 to 1.0) sets the transposition amount (-36 to 36)

"on" / "off": enables / disables transposition

Delay

value: Integer (0 to 5000) sets the amount of delay in ms,
 "on" / "off": enables / disables delay

Staccato

value: Integer (2 to 1000) sets the staccato duration in ms
 "on" / "off": enables / disables staccato

Prolong

value: Integer (0 to 2000) sets the time the note is extended after release
 "on" / "off": enables / disables the prolong function

PulseFreq (pulse frequency)

value:

- Integer (-240 to 240) sets the pulse speed;
 - o -240 to -1: 5bpm to 244bpm
 - o 0 to 240: 245ms to 5ms
- Float (-1.0 to 1.0) sets the pulse speed;
 - o -1.0 to 0.0: 5bpm to 245bpm
 - o 0.0 to 1.0: 245ms to 5ms

*Please note; in contrast to the Tremolo function (section 5.6 Utopa Tremolo), String messages like "200bpm" or "150ms" are **not** accepted by PulseFreq.*

"on" / "off": enables / disables pulse function

PulseWidth (pulse width)

value: Integer (0 to 100) or Float (0.0 to 1.0) sets the with of the pulse
 "on" / "off"

Sustain

value: Integer (0 to 100) or Float (0.0 to 1.0) sets the amount of sustain.
 "on" / "off": enables / disables the sustain function

DynWidth (dynamics width, relates to relation between velocity and valve speed)

value: Integer (0 to 100) or Float (0.0 to 1.0) sets the amount of dynamic width
 "on" / "off": enables / disables the dynamic width; *functionality unknown*

DynCenter (dynamics width, relates to relation between velocity and valve speed)

value: Integer (0 to 100) or Float (0.0 to 1.0) sets the center of the dynamic range.
 "on" / "off": enables / disables the dynamic center; *functionality unknown*

VelMin (velocity minimum)

value: Integer (0 to 100) or Float (0.0 to 1.0) sets the minimum amount of velocity to be used to make the layer sound (default = 0%)
 "on" / "off": enables / disables the velocity minimum

VelMax (velocity maximum)

value: Integer (0 to 100) or Float (0.0 to 1.0) sets the minimum amount of velocity above which the layer doesn't sound (default 100%)

"on" / "off": enables / disables the velocity maximum

Midi (MIDI output channel)

value: Integer (1 to 16) sets the output MIDI channel of the layer

"on" / "off": enables / disables the MIDI output

Examples:

```
[ "/M3/L1/Transpose", -12, "on" ] : set transposition on Manual III, layer 1 to -12 and on
[ "/M3/L1/Transpose", 0, "off" ] : set transposition on Manual III, layer 1 to 0 and off
[ "/M1/L1/PulseFreq", -100, "on" ] : set the pulse on Manual I, Layer 1 to 145BPM and on
[ "/M1/L1/PulseWidth", 50 ] : set the pulse width on Manual I, Layer 1 to 50%
[ "/M2/L2/Staccato", 20, "on" ] : set staccato on Manual II, Layer 2 to 20ms and on
```

5.6 Utopa Tremolo

The messages for controlling the Utopa **tremolos** can be composed as follows:

- Tremolo address String ("/TremOW" or "/TremHW")
- a String for the tremolo speed (optional)
- a Float for the tremolo width (optional)
- a String "on" / "off" for enabling / disabling the tremolo (optional)

[address String, speed String (optional), width Float (optional), "on"/"off" (optional)]

Each of the speed, width and "on"/"off" values can be left out, and the order of the values can also be different. The console recognizes the values by type.

The **address String** points to one of the two tremolos; /TremOW (obenwerk) applies to registers 49 to 61, and /TremHW (hauptwerk) to registers 36 to 48.

The string for tremolo **speed** should be formatted as follows:

"<amount><units>"

amount: amount (5 to 245)

units: "ms" or "bpm"

The Float for **width** ranges from 0.0 to 1.0 and sets the width percentage from 0% to 100%

The **"on" / "off"** string turns the tremolo on or off.

Examples:

```
[ "/TremOW", "100bpm", 0.3, "on" ] : set tremolo OW to 100bpm, width 30% and on
[ "/TremHW", "200ms", 0.7, "on" ] : set tremolo HW to 200ms, width 70% and on
[ "/TremHW", "off" ] : turn tremolo HW off
```

5.7 Swell pedals

The swell pedals are numbered from 0 to 3 in the OSC Implementation. The message to set their value can be composed as follows:

- an address String (pointing to the pedal)
- a value (Float)

[address String, value (Float)]

The **address String** is formatted as follows:

"SP<pedal>"

pedal: the number of the pedal;

- 0: Wind motor Utopa
- 1: Crescendo pedal
- 2: Swell box Sauer
- 3: (unknown/currently unassigned)

The **value** should be a Float ranging from 0.0 to 1.0

Examples:

["/SP0", 1.0] : set wind for Utopa to maximum

["/SP0", 0.5] : set wind for Utopa to halfways

["/SP2", 0.0] : close the swell box of the Sauer organ

5.8 Setzer

In the current implementation stored Setzer setting be recalled via OSC. There are commands for:

- the next slot
- the previous slot
- calling an individual slot

Storing Setzer data is not possible via OSC.

[address String, command String, value String]

The **address String** is as follows:

"/Setzer"

The **command String** can be:

"next"

"previous"

"call"

The **value String** is only used for the *“call”* command. It is composed as follows:

“<bank><letter><index>0”

bank : the bank number, always three digits between 001 and 999

letter: the letter, in capital (A-E)

index: the index, always two digits between 01 and 10

Note that after the index number there is always an extra “0”. The bank number can be anything below 999, but experience learns that higher bank numbers only get called when they actually have a preset stored in them, and it is actually very hard from the organ interface itself to reach a bank number over 100. With *“next”* and *“previous”* it is also possible to go to empty presets. Please also be aware that the OSC functionality of the Setzer currently only features recall of settings and doesn’t have options to store anything.

Examples:

[*“/Setzer”, “call”, “001A020”*] : call preset 1 A 2

[*“/Setzer”, “call”, “010C100”*] : call preset 10 C 10

[*“/Setzer”, “next”*] : call next preset

[*“/Setzer”, “previous”*] : call previous preset

5.9 Single commands

There are a few single commands. A single command only consists of an address String, there are no additional values.

[*address String*]

The **address String** for **creating layers** is composed as follows:

"/M<manual>/createLayer"

<manual>: manual number 0,1,2,3, where 0 means the Pedal section

There is also a command for **resetting the console**. When used, all registers are turned off in all layers, all layer settings are set to default values, all extra layers are removed (so that only the original 4 layers per manual remain) and all currently playing notes are stopped. The **address String** for this command is as follows:

"/panic"

Examples:

[*"/M3/createLayer"*] : add a layer to Manual III

[*"/panic"*] : reset everything and stop sounding notes

6 Console MIDI implementation

The digital console for Utopa and Sauer organs can be controlled via MIDI. In fact, the console itself *is* a MIDI controller. Each of the manuals, pedals, knobs and keys sends out a MIDI signal to the computer inside the console, controlling all of its functionality. The MIDI implementation of the console is fully created with this purpose. If you want to control the device via external MIDI, you will need to send the same MIDI messages as the console itself does, and the software makes no distinction to where the MIDI comes from. I.e. it acts as if the buttons, keys and knobs were pushed on the console itself. As explained in section 3 How to control the organ, this is good if you just want to play notes. And not so good if you want to access any of the other functionality of the machine, unless you know very well what you are doing. *For controlling registers and settings on the device I strongly recommend using OSC instead*, but, if you insist, below is a list of the MIDI messages the device understands.

6.1 Notes

The manuals and pedal section of the digital console each have their own MIDI channel. Notes can be played by sending MIDI **note on / note off** to the corresponding channel:

MIDI Channel 1: Pedals
 MIDI Channel 2: Manual I
 MIDI Channel 3: Manual II
 MIDI Channel 4: Manual III

The console also understands velocity values, and can for example switch layers on and off via velocity.

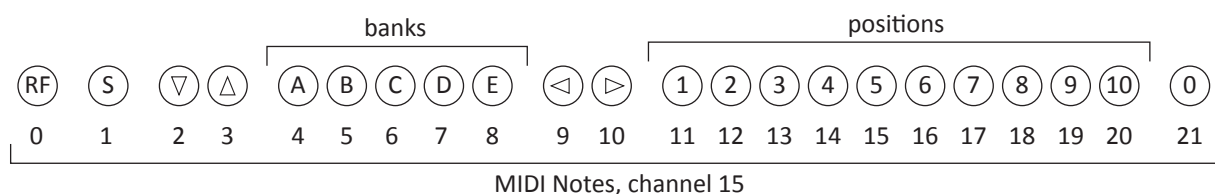
6.2 Swell pedals

The swell pedals of the digital console can be controlled via **MIDI control** messages (cc).

MIDI Channel 2, Control 1 (modulation): Utopa wind motor
 MIDI Channel 4, Control 1 (modulation): Sauer wind chest

6.3 Setzer

The only way to operate the organ's Setzer system is to trigger the buttons for the Setzer directly via **MIDI note on / off** messages. These messages are on **MIDI Channel 15** (counting from 1-16), and sending a message has the same effect as pressing the physical button on the console (MIDI note 0 = C-2).



6.4 "The rest"

All other functionality can only be controlled by directly sending the same messages as the console does itself when touching knobs or hitting keys. As knobs and buttons can change function according to context it is hard to know what you actually control when sending these messages yourself. But if you want to try, here is a list. Beware that the physical register switches *will not flip* when the message is sent. MIDI channels count from 1-16.

Channel	Note	Description			
14	0		14	57	58 Quinta 1 1/3'
14	1	1 Dulciana 8'	14	58	59 Su_fflött 1'
14	2	2 Bourdon 8'	14	59	60 Scharff IV
14	3	3 Flute harmonique 8'	14	60	61 Vox humana 8'
14	4	4 Gamba 8'	14	61	62 Principal 16'
14	5	5 Principal 8'	14	62	63 Subbass 16'
14	6	6 Rohrflöte 4'	14	63	64 Quintbass 12'
14	7	7 Octave 4'	14	64	65 Octav 8'
14	8	8 Bourdon 16'	14	65	66 Posaune 16'
14	9	9 Rauschquinte 2 2/3' 2'	14	66	67 Posaune 8'
14	10	10 Cornett-Mixtur 3-5fach	14	67	68 Clarin 4'
14	11	11 Trompete 8'	14	68	1 sauer grey
14	12	12 Schwellgedeckt 16'	14	69	2 sauer green
14	13	13 Subbass 16'	14	70	3 sauer white
14	14	14 Cello 8'	14	71	4 sauer blue
14	15	15 Bassflöte 8'	14	72	5 sauer yellow
14	16	16 Octavbass 8'	14	73	6 sauer pink
14	17	17 Contrabass 16'	14	74	16
14	18	18 Contrabass 32'	14	75	8
14	19	19 Posaune 16'	14	76	4
14	20	20 Posaune 32'	14	77	chord
14	21	21 Motor ab	14	78	bass
14	22	22 Quintatone 8'	14	79	melody
14	23	23 Konzertflöte 8'			
14	24	24 Voix celeste 8'	14	80	Layer Open P.1
14	25	25 Violine 4'	14	81	Layer Open P.2
14	26	26 Traversflöte 4'	14	82	Layer Open P.3
14	27	27 Flautino 2'	14	83	Layer Open P.4
14	28	28 Harmonia aeterea 3fach	14	84	Layer Open I.1
14	29	29 Tremulant	14	85	Layer Open I.2
14	30	30 Aeoline 8'	14	86	Layer Open I.3
14	31	31 Fugara 8'	14	87	Layer Open I.4
14	32	32 Flötenprincipal 8'	14	88	Layer Open II.1
14	33	33 Lieblich Gedeckt 16'	14	89	Layer Open II.2
14	34	34 Oboe 8'	14	90	Layer Open II.3
14	35	35 Clarinette 8'	14	91	Layer Open II.4
			14	92	Layer Open III.1
14	36	36 Burdon 16'	14	93	Layer Open III.2
14	37	37 Rohrflöte 8'	14	94	Layer Open III.3
14	38	38 Quintatone 8'	14	95	Layer Open III.4
14	39	39 Octav 4'			
14	40	40 Gemshorn 4'	14	96	Layer Mute P.1
14	41	41 Weitpfeife 2'	14	97	Layer Mute P.2
14	42	42 Sexquintaltra II	14	98	Layer Mute P.3
14	43	43 Mixtur V	14	99	Layer Mute P.4
14	44	44 Cymbel III	14	100	Layer Mute I.1
14	45	45 Cornett V	14	101	Layer Mute I.2
14	46	46 Fagott 16'	14	102	Layer Mute I.3
14	47	47 Trompet 8'	14	103	Layer Mute I.4
14	48	48 Gedackt 8'	14	104	Layer Mute II.1
14	49	49 Gedackt 8'	14	105	Layer Mute II.2
14	50	50 Violdigamba 8'	14	106	Layer Mute II.3
14	51	51 Unda maris 8'	14	107	Layer Mute II.4
14	52	52 Principal 4'	14	108	Layer Mute III.1
14	53	53 Rohrflöte 4'	14	109	Layer Mute III.2
14	54	54 Nassat 3'	14	110	Layer Mute III.3
14	55	55 Octav 2'	14	111	Layer Mute III.4
14	56	56 Waldflöte 2'	14	112	Layer Sustain P.1
14	57	57 Tertia 1 3/5'	14	113	Layer Sustain P.2

Interface Manual / 6 Console MIDI implementation

14	114	Layer Sustain	P.3	15	58	Encoder (cap press)	3
14	115	Layer Sustain	P.4	15	59	Encoder (cap press)	4
14	116	Layer Sustain	I.1	15	60	Encoder (cap press)	5
14	117	Layer Sustain	I.2	15	61	Encoder (cap press)	6
14	118	Layer Sustain	I.3	15	62	Encoder (cap press)	7
14	119	Layer Sustain	I.4	15	63	Encoder (cap press)	8
14	120	Layer Sustain	II.1	15	64	Encoder (cap press)	9
14	121	Layer Sustain	II.2	15	65	Encoder (cap press)	10
14	122	Layer Sustain	II.3	15	66	Encoder (cap press)	11
14	123	Layer Sustain	II.4	15	67	Encoder (cap press)	12
14	124	Layer Sustain	III.1	15	68	Encoder (cap press)	13
14	125	Layer Sustain	III.2	15	69	Encoder (cap press)	14
14	126	Layer Sustain	III.3	15	70	Encoder (cap press)	15
14	127	Layer Sustain	III.4	15	71	Encoder (cap press)	16
15	0	Setzerleiste	RF	15	72	Encoder (cap press)	17
15	1	Setzerleiste	S	15	73	Encoder (cap press)	18
15	2	Setzerleiste	E-	15	74	Encoder (cap press)	19
15	3	Setzerleiste	E+	15	75	Encoder (cap press)	20
15	4	Setzerleiste	A	15	76	Encoder (cap press)	21
15	5	Setzerleiste	B	15	77	Encoder (cap press)	22
15	6	Setzerleiste	C	15	78	Encoder (cap press)	23
15	7	Setzerleiste	D	15	79	Encoder (cap press)	24
15	8	Setzerleiste	E	15	80	Encoder (cap press)	25
15	9	Setzerleiste	<	15	81	Encoder (cap press)	26
15	10	Setzerleiste	>	15	82	Encoder (cap press)	27
15	11	Setzerleiste	1	15	83	Encoder (cap press)	28
15	12	Setzerleiste	2	15	84	Encoder (cap press)	29
15	13	Setzerleiste	3	15	85	Encoder (cap press)	30
15	14	Setzerleiste	4	15	86	Encoder (cap press)	31
15	15	Setzerleiste	5	15	87	Encoder (cap press)	32
15	16	Setzerleiste	6	15	88	Encoder (cap press)	33
15	17	Setzerleiste	7	15	89	Encoder (cap press)	34
15	18	Setzerleiste	8	15	90	Encoder (cap press)	35
15	19	Setzerleiste	9	15	91	Encoder (cap press)	36
15	20	Setzerleiste	10	15	92	Encoder (cap press)	37
15	21	Setzerleiste	0	15	93	Encoder (cap press)	38
15	24	new Layer		15	94	Encoder (cap press)	39
15	25	OK		15	95	Encoder (cap press)	40
15	26	edit		15	96	Encoder (cap press)	41
15	27	ESC		15	97	Encoder (cap press)	42
15	30	<		15	98	Encoder (cap press)	43
15	31	>		15	99	Encoder (cap press)	44
15	32	ins		15	100	Encoder (cap press)	45
15	33	del		15	101	Encoder (cap press)	46
15	34	cpy		15	102	Encoder (cap press)	47
15	35	pst		15	103	Encoder (cap press)	48
15	36	Crescendi	1	15	104	Encoder (cap press)	49
15	37	Crescendi	2	15	105	Encoder (cap press)	50
15	38	Crescendi	3	15	106	Encoder (cap press)	51
15	39	Crescendi	4	15	107	Encoder (cap press)	52
15	40	on/off		15	108	Encoder (cap press)	53
15	41	n.c.		15	109	Encoder (cap press)	54
15	42	>		15	110	Encoder (cap press)	55
15	43	pedal left	Loops	15	112	Encoder (spin logic)	1
15	44	pedal right	Loops	15	113	Encoder (spin logic)	b
15	45	First manual left	Loops	15	114	Encoder (spin logic)	2
15	46	First manual right	Loops	15	115	Encoder (spin logic)	b
15	47			15	116	Encoder (spin logic)	3
15	48	<		15	117	Encoder (spin logic)	b
15	49	>		15	118	Encoder (spin logic)	4
15	50	Midirecorder	rec	15	119	Encoder (spin logic)	b
15	51		ply	15	120	Encoder (spin logic)	5
15	52		stp	15	121	Encoder (spin logic)	b
15	53		rew	15	122	Encoder (spin logic)	6
15	54		fwd	15	123	Encoder (spin logic)	b
15	56	Encoder (cap press)	1	15	124	Encoder (spin logic)	7
15	57	Encoder (cap press)	2	15	125	Encoder (spin logic)	b
				15	126	Encoder (spin logic)	8
				15	127	Encoder (spin logic)	b

7 List of registers and ranges

Each register of the instruments has a specific name and range. As reference, here follows a list of the registers, their ranges, and, if applicable, amount of transposition in semitones of the sounding pipes relative to 8 foot. This list is also provided separately from the manual, in pdf, csv and excel formats, as well as visual representations of the (transposed) ranges.

#	name	type	organ	section	lowest	highest	transposition
1	Dulciana	8	Sauer	D3	36	91	0
2	Bourdon	8	Sauer	D4	36	91	0
3	Flute harmonique	8	Sauer	D3	36	91	0
4	Gamba	8	Sauer	D3	36	91	0
5	Principal	8	Sauer	D4	36	91	0
6	Rohrflote	4	Sauer	D4	36	91	12
7	Octave	4	Sauer	D3	36	91	12
8	Bourdon	16	Sauer	D3	36	91	-12
9	Rausch-quinte	2 2/3, 2	Sauer	D4	36	91	
10	Cornett-mixtur	3-5 fach	Sauer	D4	36	91	
11	Trompete	8	Sauer	D3	36	91	0
12	Schwell-gedeckt	16	Sauer	D1	36	65	-12
13	Subbass	16	Sauer	D1	36	65	-12
14	Contra-bass	16	Sauer	D2	36	65	-12
15	Cello	8	Sauer	D1	36	65	0
16	Bassflote	8	Sauer	D1	36	65	0
17	Octav-bass	8	Sauer	D1	36	65	0
18	Contra-bass	32	Sauer	D2	36	65	-24
19	Posaune	16	Sauer	D2	36	65	-12
20	Posaune	32	Sauer	D2	36	65	-24
21	Motor ab		Sauer	D3			
22	Quintaton	8	Sauer	D5	36	103	0
23	Konzert-flote	8	Sauer	D5	36	103	0
24	Voix celeste	8	Sauer	D5	48	103	0
25	Violine	4	Sauer	D5	36	103	12
26	Travers-flote	4	Sauer	D5	36	103	12
27	Flautino	2	Sauer	D5	36	91	24
28	Harmonia aetheria	3 fach	Sauer	D5	36	91	
29	Tremulant		Sauer	D5			
30	Aeoline	8	Sauer	D6	36	103	0
31	Fugara	8	Sauer	D6	36	103	0
32	Floten-principal	8	Sauer	D6	36	103	0
33	Lieblich Gedeckt	16	Sauer	D6	36	103	-12
34	Oboe	8	Sauer	D6	36	103	0
35	Clarinetten	8	Sauer	D6	36	103	0

Interface Manual / 7 List of registers and ranges

36	Burdun	16	Utopa	HW	36	92	-12
37	Principal	8	Utopa	HW	36	92	0
38	Rohrflott	8	Utopa	HW	36	92	0
39	Quintathen	8	Utopa	HW	36	92	0
40	Octav	4	Utopa	HW	36	92	12
41	Gemshorn	4	Utopa	HW	36	92	12
42	Weit Pfeiffe	2	Utopa	HW	36	92	24
43	Sexquint altra	2 fach	Utopa	HW	36	92	
44	Mixtur	5 fach	Utopa	HW	36	92	
45	Cymbel	3 fach	Utopa	HW	36	92	
46	Cornett	4 fach	Utopa	HW	59	92	
47	Fagott	16	Utopa	HW	36	92	-12
48	Trompet	8	Utopa	HW	36	92	0
49	Gedackt	8	Utopa	OW	36	92	0
50	Violdi-gamba	8	Utopa	OW	36	92	0
51	Unda maris	8	Utopa	OW	56	92	0
52	Principal	4	Utopa	OW	36	92	12
53	Rohrflott	4	Utopa	OW	36	92	12
54	Nassat	3	Utopa	OW	36	92	19
55	Octav	2	Utopa	OW	36	92	24
56	Waldflott	2	Utopa	OW	36	92	24
57	Tertia	1 3/5	Utopa	OW	36	90	28
58	Quinta	1 1/2	Utopa	OW	36	92	31
59	Sufflott	1	Utopa	OW	36	92	36
60	Sharf	4 fach	Utopa	OW	36	92	
61	Vox humana	8	Utopa	OW	36	92	0
62	Principal	16	Utopa	P	36	66	-12
63	Subbass	16	Utopa	P	36	66	-12
64	Quint Bass	12	Utopa	P	36	66	-5
65	Octav	8	Utopa	P	36	66	0
66	Posaune	16	Utopa	P	36	66	-12
67	Posaune	8	Utopa	P	36	66	0
68	Clarin	4	Utopa	P	36	66	12