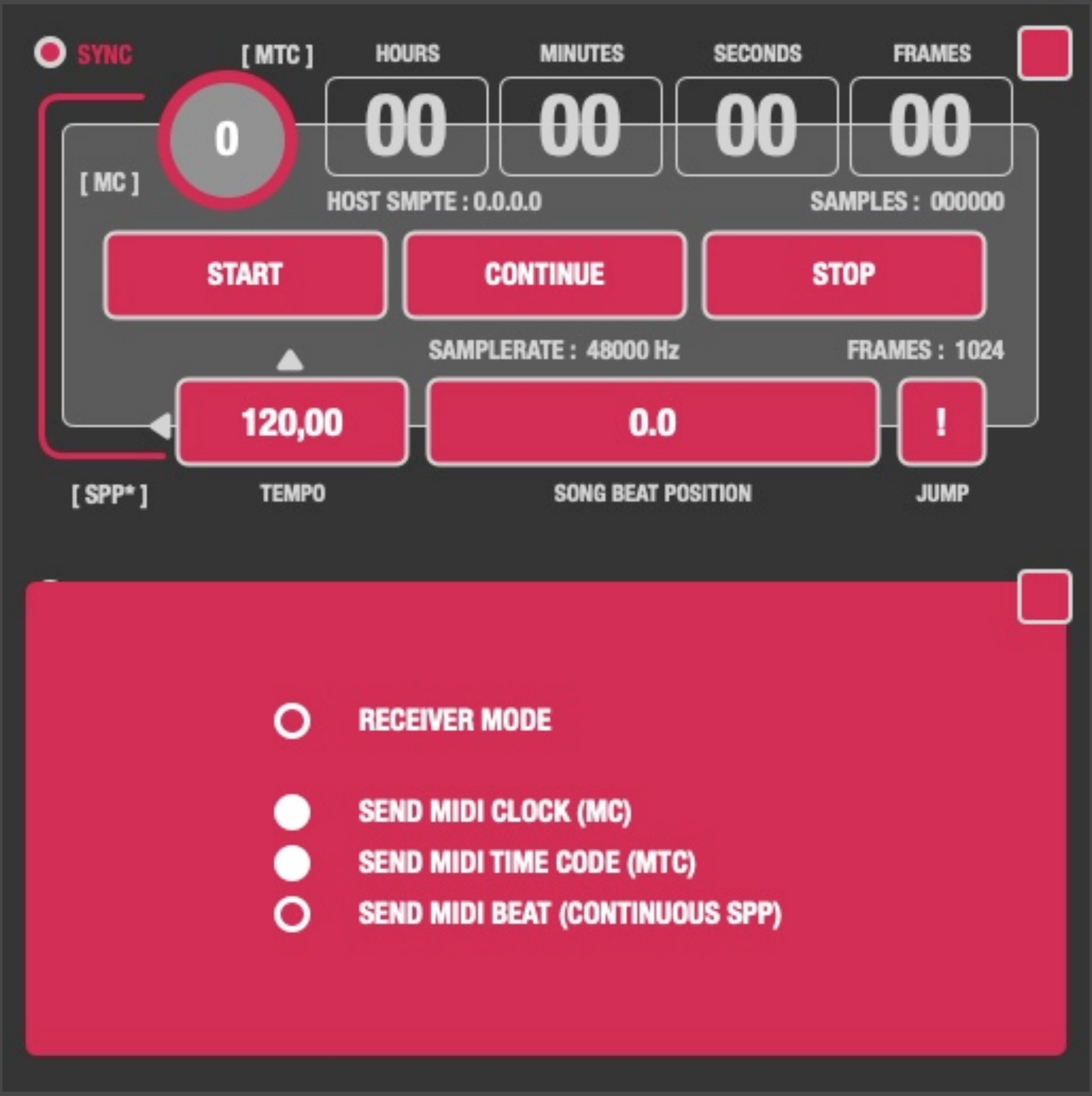
MIDI SWEET SERIES

MIDI SWEET : MIDI Time Control [AUDIO UNIT]

MANUAL

Create realtime MTC / MC / SPP signals with audio clock precision for transmission via MIDI



○ RECEIVER MODE

● SEND MIDI CLOCK (MC)
● SEND MIDI TIME CODE (MTC)
○ SEND MIDI BEAT (CONTINUOUS SPP)

In Short

MIDI Time Control is a MIDI enabled AudioUnit (instrument), which is specialized for generating MIDI Time Code (MTC), MIDI Clock (MC) and Song Position Pointer (SPP*) signals, optionally altogether, for transmission via the standard MIDI protocol inside or for sending outside an audio unit system.

The unit generates all signals with audio clock precision, and is optionally independent of the hosts provided synchronisation mechanisms.

The unit will not make any sound at all, but merely generate, send and receive MIDI messages based on user configuration and interaction. An audio unit host with active MIDI callback and routing support is required for useful operation.

MIDI Time Control also can de-jitter incoming signals and virtually divide a clock signal by a tempo factor. This is done with precise audio unit host synchronisation rather than with incoming MIDI clock translation.

I seems to be a quite simple device at first view, but in fact, such audio unit is internally extremely complex and required 1 year of continuous development and research for the entire topic on our side. So this is reflected in the price of the unit. It does not use any external MIDI, audio or graphics frameworks for some certain reasons. Everything was developed from scratch here, as with all of our unique audio unit developments.

SIDENOTE: It is not recommended to run multiple synchronisation signals from different sources (i.e. a host signal plus an audio unit generated signal altogether). This may cause confusion on the receivers end. Only one master sync signal (including MC, MTC, SPP) should be sent to a device at a time and from a single source.

MIDI Time Control (“MIDI Clockworks”)

MIDI Time Control will produce absolutely synchronous, sample accurate events inside an audio unit processing environment, there is no dependency from any timers or threads other than the one and only stable master audio clock of the systems running audio driver. The signal is generated inside the audio kernel.

So audio and MIDI can be synchronized 100% perfectly, if everything is calculated and transmitted correctly inside a closed digital rendering system.

Remarks: The audio unit framework will usually process blocks of samples and its MIDI implementation ensures, that exact offsets are calculated into these sample buffers. So the results of MIDI generation are by design sample accurate with the precision of one sample (the sample rate frequency itself). The generation of all midi signals occurs directly inside the core audio kernel thread and the user interface is completely decoupled from the core processing.

However, if the generated MIDI messages are sent outside by the host for transmission to external devices, the accuracy of the timing very much depends on the hosts implementation of the transmission protocols and the used OS dependencies for such.

So in case of external transmission, there may be introduced some unavoidable amount of signal delaying and jittering due to external processing and transmission issues, especially if bluetooth or ethernet connections are actively used for MIDI transmission outside the running master clock system.

MTC - The MIDI Time Code

MIDI Time Code is one of the standard MIDI system common messages, which was developed for encoding the real world timeline into continuous MIDI events. It has many in common with the classic SMPTE signal generation and essentially will provide such signal.

There are 2 different encoding schemes for MIDI Time Code messages. The first is widely called “quarter frame messages” and basically a sequence of 8 following short 2-bye messages, which are cyclic and in sum build up a complete MIDI time code in the form of “00:00:00:00”. The result is an expression of HOURS : MINUTES : SECONDS : FRAMES, which cyclically repeats at frame rate. We use 24 frames per second.

It is like a stop watch, which can be started at any point in time and will draw a picture of the real world timeline via MIDI. This kind is used for realtime progression and we will use this format for the running state actively.

The second method of encoding such MTC message is using SysEx In form of a fixed length sequence of bytes (i.e. “0xF0 0x7F 0x7F 0x01 0x01 [HH] [MM] [SS] [FF] 0xF7”, whereby HH stands for hours, MM for minutes, SS for seconds and FF for frames). Frames are a kind of artificial subdivisions of a second, instead using usual milliseconds. This all mainly comes from video production, where “frames” are indeed referencing the picture frame play rate. ~~We are using SysEx for transmission only while jumping to a certain discrete time frame with the manual mode, not for the continuous realtime generation.~~ Remarks: SysEx messages are not well supported by most audio units.

There also exist several different defined possible frame rates, but we will use exclusively 24 frames per second. The reason is mainly, that the frame rate of 24 corresponds to the MIDI Clock resolution, which is discussed later in detail.

Remember: MIDI Time Code is capturing the real world time, which is constant and therefore no tempo information will be encoded with it. This is an often mismatched fact by users. For transmission of tempo events there are other methods offered by the MIDI standard and we will use these.

The real world time can be described as a constant spaced grid of time, which never ever will react to or reflect any musical tempo changes and never will affected by sample rate changes or anything like that.

Our MTC has a limit of the maximum generated time code with the value 23:59:59:23 (a full day). After this, it will stop and must be started from zero. This conforms to the MIDI standard.

MC - MIDI Clock

MIDI Clock was developed for encoding and transmitting tempo information exclusively. MTC and MC signals can be sent together but are fundamentally different, so giving both, a real world timeline (constant) and a tempo information (dynamic) at any current time point with a MIDI stream. The frame repetition rates are independent with both signals. MTC is constant in time, MC will change according the given tempo. With such combination, basically perfect synchronisation of musical events can be achieved.

The MIDI Clock signals are 1byte pulses, that have no other information than the (variable) time span between them. A receiver must decode the tempo information by continuously capturing the time between such consecutive events. So this decoding process is basically delayed by a small amount of time.

On digital systems this is easily done by counting the exact sample frames between the received impulses. With the current sample rate, a discrete value then can be calculated, which is corresponding to a certain tempo value respectively.

This way, all receiving devices can calculate the current tempo in nearly realtime and adopt it into their signal processing.

Note: This kind of tempo information is usually not required by the AudioUnit System, as there is a direct function (if even implemented) to the hosts tempo information. However it has its meaning, if trying to synchronize external devices.

The MC signal is generally sent with a resolution of 24 frames per quarter note, independently of any musical division, only driven by the tempo. This is a standard MIDI resolution of 96 pulses per 4/4 bar or whole note. Sometimes it is called MIDI Beat or MIDI Beat Clock. Please note that this is not corresponding to the common meaning of the term “beat”. A so-called MIDI beat will occur 96 times inside a 4/4 bar of a song or 48 times per second at the exact tempo of 120 beats per minute.

We will use the term “beat” here exclusively in the sense of real beats later in the SPP description.

MIDI START, STOP, CONTINUE

There are some important MIDI status messages, which can be seen in direct relation to the MC and MTC signals. This are the quite self explaining commands Start, Stop and Continue.

These messages are also 1byte long and will usually start the generation of MC pulsing. In our implementation, the MTC, SPP and the MC generation are side by side events. If such commands are triggered, all enabled pulse generators will react equally to it and accordingly start, pause and stop their pulsing, along with our optional SPP* beat counter position.

MIDI Time Control can send and even receive START, STOP and CONTINUE commands, which is dependent of the chosen operation mode.

If triggered via the user interface of MIDI Time Control, the messages will be sent out over MIDI together with some positioning information discussed later. This way it is possible to remote control external apps and devices if these even understand such commands. But we cannot give any guarantees, as many devices use quite different schemes of MIDI implementation regarding that and may produce quirks in some situations.

SPP - Song position Pointer

Repeating: The sending of MIDI impulses for the current tempo does not send any other information than itself. The pulses must be captured by the receiver to gain a tempo information. There is a certain problem with this, which is: a completely missing positioning information. The clock signal is independent of any time or beat position information.

For positioning, there was another MIDI standard developed, the Song Position Pointer. SPP is an absolute song position event in fractions of a beat (16th note) with a max. resolution of 6 frames or pulses. (Remember: a quarter (or “real beat”) corresponds to 24 frames - the MIDI clock rate, a whole note, four on the flor, respectively corresponds to 96 frames then.)

The SPP is tempo and timecode independent, just another grid, which now works consequently tempo changes and can be placed at any time code position and also independent of sending any clock signal.

The Song Position Pointer indicates concrete beat position information with a resolution of a 16th note. In difference to the MC pulse, which is sent continuously 24 times per quarter and our MTC, also sent at this frame rate but with rather fixed time intervals aligned to the current sample rate, the SPP is **usually not sent continuously but ones, between Stop and Continue** commands.

It is completely independent of the MTC but often works in conjunction with the MC signal.

Please note, that the song position pointer always starts with zero (0) and not with 1. It also has technically a limited resolution of max. 16384 possible values for 16th steps. This results in a maximum of 4096 full beat positions (quarters). So the maximum song position is limited and the resulting song length very much dependent from the current tempo. With a standard tempo of 120 BPM this will result in exactly 34 minutes and 8 seconds for the maximum song length.

If a SPP is received, the device usually will stop and jump to the focused beat position with the 16th note resolution and then continue (per consecutive “continue” command) from there, with the transmitted tempo information by the following clock (MC) signal again.

This means that our optional **continuous transmission** of the SPP signal is highly experimental and potentially could confuse some receiving devices, as these possibly repeatedly try to jump to the transmitted beat positions, by stopping and (expecting) continuing. This can cause some unwanted scattering. There is the option to switch the continuous beat position generation off and exclusively using manual adjustment, which by the way, can be automated with the exposed audio unit parameters.

Some devices will even only work correctly, if there is a concrete sequence of STOP, SPP, CONTINUE, repeating only at certain points of the song (i.e. with loops) and in stopped mode. They will generally stop if a SPP is received and wait for a continue command. Note: Our SPP generally will not take any host submitted loop points into account for certain reasons.

However we have built in a special option, that is able to generate continuous SPP signals in form of a commonly used “Beat Position Counter”, a floating point value, which allows to transmit the current beat position at quarter note resolution (“four on the floor”), independently from any musical division across the entire song progression.

This beat position will be transmitted via MIDI merely, if it actually fits into the maximum MIDI range of 4096 possible beats and it will be transmitted at whole beat position. After this, the counter still may increase on the interface or will become updated to its value with manual time and tempo adjustments, but there will no outgoing MIDI command generated for it.

A beat position can be used to jump and follow to a concrete beat in the song, even independent of all tempo changes that were made prior. The beat grid is basically tempo independent, meaning it always will adopt any tempo. So it is fundamentally different than the MIDI Time Code, which is definitively tempo independent and will not change at any tempo change inside the time line and its progression. This time is essentially undefined at any given time point in a real time system if dynamic realtime tempo changes occur during playback.

Remarks: Apple also uses the concept of a continuous beat position pointer across their AVAudioSequencer class and the entire MusicSequence framework, but these values are not transmitted as any MIDI data but accessible directly with the MusicSequence and AudioUnit framework functions. i.e with the “musicalContext” block.

MIDI SPP is technically spoken a 2byte MIDI common message with its own status byte and will encode the song beat positions in 16th note resolution with a 14bit value analog to the MIDI pitch bend controller (core and fine adjustment). In sum 3 bytes are transmitted with such a message.

A value of 0x08 means 8 16th notes steps, resulting in 48 completed clock pulse width or 2/4 notes distance. The new position is then right at the start position of the 3rd quarter note.

So our “beat position counter” uses the SPP encoding scheme for continuously transmitting the current beat position in quarter notes resolution and it additionally uses the continuous clock signal to track any tempo changes in realtime. If a receiver will decode such pulsed signals, it actually would be able to exactly follow the current play position of the sender and the current tempo, which basically can result in nearly perfect app and device synchronisation. At least in theory. The MIDI time code can add to this by showing the absolute time elapse.

We will not send stop, and continue commands before and after such consecutive continuous SPP* commands in running mode. However, we will send such sequences with all manual adjustments in transmitter mode.

In manual transmitter mode, the sent SPP command will just stop and jump to a certain position, if the receiver supports it. Some receivers will ignore jump commands at all, some other merely while a clock signal is not in running state and will therefore only react to it, if the clock has actually paused or stopped. So the manual mode expects explicit “start”, “stop” and “continue” commands to operate complete remote transport.

The MIDI standards regarding SPP, offer wide range of possible implementations. It cannot be guaranteed, that our MIDI Time Control will work under all possible circumstances. There may be side effects with certain devices and apps. But potentially it could be a quite perfect solution for solid inter-device and app synchronisation.

The manual transmitter mode allows to navigate thru a timeline with commands and also jump to concrete beat or time positions in a song. It is even possible with audio unit parameter automation.

Excurse: In MIDI sequencers the beat position is usually displayed independently of tempo changes, unlike the real world timeline, which must be recalculated each time a tempo change occurs, according to a kind of tempo map. So changing tempi will either stretch the graphical beat position scaling or the real world timeline. In a realtime system this is not possible due to a missing pre-defined tempo map.

Receiver Or Transmitter?

Our unit for MC, MTC and SPP can operate in receiver or transmitter mode. Default is always the transmitter mode. The receiver mode was mainly implemented for diagnostic purposes. So it is possible to check externally received signals or validate the correctness of generated signals with an extra instance.

In receiver mode, the unit is listening for incoming MIDI messages regarding MC, MTC and SPP all at ones. These can come from other apps and audio units or external devices, correctly connected to the MIDI input port of the audio unit. In this mode all controls for adjustment are disabled, as these do not make sense here.

Note: In receiver mode only quarter frame messages are decoded to MIDI Time Code, SysEx is not supported. The tempo is calculated by capturing the timeframes of the MIDI clock, the SPP is decoded as fractional beat position from the input.

The MIDI received signals will be decoded and displayed on the interface accordingly and the MIDI stream is normally passed thru to the units output MIDI port, so that it can be used further by other audio units.

BUT the options for sending MIDI generated signals out **must be disabled for sending MIDI THRU**!

If any of the options for sending MIDI signals out is enabled in receiver mode, the unit will replace all incoming signals with its internally generated signals, which for instance enables **de-jittering of incoming signals**. In this case, only the decoded tempo information will be used to generate a completely new internal sample accurate output for the received MC, MTC and SPP, which are all optional for sending out.

The receiver mode with enabled send signals is essentially operating as a re-syncing master clock source with a kind of remote input.

So do not enable any MIDI send, if you just want to analyse and pass the incoming signal thru unchanged.

How Jumping To A Position

if you want to jump to a certain position with a connected remote device, you must have enabled the transmitter mode. Then adjust the Beat Counter display by sliding into any direction and tap the jump button or use continue. MIDI Time Control will then send the Song Position Pointer and MIDI Time Code that is currently displayed and update all the depending displays accordingly.

The controls for tempo and beat position have an inbuilt fine control. The left side will adjust coarse values, the right side will adjust fine values.

~~In transmitter mode, adjusting the MTC displays for hours, minutes, seconds and frames (independently) will generate MTC full MTC SysEx messages, which allow to effectively navigate to a certain position in an imaginary timeline. Such messages are widely used for stop mode navigation. The beat counter display will be updated with this too.~~

Remarks: The automatic calculations in MIDI Time Control assume a constant tempo. It is just impossible to calculate any relative tempo changes prior jumping to a specific virtual location. There is no tempo map like in a MIDI sequencer that could be read and preprocessed. So the MTC displays of a connected sequencer can widely differ in time, as there may be internal calculated tempo maps inside the external sequencer.

MIDI Time Control is a realtime generator. So the beat counter and timecode displays will be calculated (synchronized) based on the assumption of a constant tempo, which is the currently displayed tempo value on the tempo display.

Remote Control With Audio Unit Parameters

Certain roles for audio unit parameter automation do apply. For any manual adjustment or parameter automation, the mode must be switched to transmitter mode. Other modes do have no effect to parameter automation at all.

You also must usually proceed certain sequences of commands. To jump to a position for instance, adjusting hours, minutes, seconds, frames or beats as desired requires a “stop” command, then requiring a “continue” command to proceed. By adjusting one of the positioning controls or parameters, the the new position will be sent to MIDI out. The “continue” command must be triggered afterwards for proceeding with the playback.

Special roles apply to the Start, Stop and Continue parameters. These parameters are conceptually realtime **triggers**, but have parameters for automation support. This means, that the value will be set to zero internally again after a trigger occurred. This is especially important if automating parameters with external graphical editing tools and sequencers. It may produce unexpected results, if these triggers are automated wrongly or continuously for instance. or just stay at value 1.0 (on) for a long time.

With MIDI automation (i.e. by assigning MIDI controllers to the audio unit parameters), complex synchronisation projects can be achieved but requires some care, trial and a certain plan. In transmitter mode, incoming start, stop and continue commands are also supported but users should prefer audio unit parameter automation and not mix up everything.

Host Synchronisation

Generally it is (by conception) thought to operate with MIDI Time Control as a master effect, independently of the hosts transport or tempo control. It even can be remote controlled with the host via parameter automation and some MIDI commands (for start, stop and continue).

So MIDI Time Control is basically a master device. However we implemented the to-host sync option, which enables for instance to **provide a clock divider simulation** by a fractional tempo division factor of the hosts tempo. This feature is often asked by users.

This is not a real “clock divider” mechanism, as this requires to implement an independent receiver and a transmitter at same time, while the receiver merely decodes the tempo and the transmitter recalculates an outgoing signal with new, precise timing. NOTE: Exactly this mechanism can effectively be achieved in receiver mode and any enabled send MIDI out signals with MIDI Time Control, which acts as a signal de-jittering device then - the new output signal is generated sample accurately based on the internally synced transmitter to the receivers input signal.

There is a certain drawback with host sync in general. The audio unit system unfortunately has no mechanism to receive sample accurate tempo and positioning information from a host. Everything must be called up optionally by the audio units at (reduced) block size rate by a call to the provided callback functions. So this may introduce small latencies and a certain amount of minimal asynchrony. The units generation usually will be somewhat delayed, but constant in its internal accuracy then afterwards, assuming the host isn’t continuously reporting crazy jumping values, as some hosts actually do!

This finally means, that adjusting a sample accurate offset for instance is just not possible for an audio unit with host sync activated, because there is no sample accurate event, that could indicate a tempo change or a transport change event by the host. This is an audio unit system flaw by design.

However, with the parameter and MIDI control, MIDI Time Control can always be automated sample accurately in its normal operation mode as a master (without the annoying host sync options). It just requires a timeline editor for audio unit parameters with sample accurate events.

If the sync option is chosen, then the hosts current tempo will be adopted and the tempo parameter will be replaced by a new tempo factor parameter around the normalized 1.0 value. A tempo factor of 0.5 will half the tempo, a factor of 2.0 will then double the tempo vice versa. Any fraction between 0.25 and 4.0 is possible. The tempo itself cannot be adjusted directly in sync mode with the unit anymore, merely with the host controls then. This is the meaning of “sync”.

The unit also will try to react to any transport change of the host and accordingly update the state, but looping is not supported. The generation of outgoing signals will be done by the unit exclusively, not by the host mechanisms in sync mode. This kind of synchronisation still will leave some gaps due to the inconsistent AudioUnit framework implementation.

Remarks: Hosts looping markers are not supported by the unit by now, as these are too complex and differently handled by hosts for a save implementation into an audio unit of such kind. Some hosts do not provide useful transport information at all or transmit values, that are completely out of any logic.

Please note, that the time resolution of the MTC signal generally cannot be changed with any tempo change, as it relies on the current sample rate and always **must reflect the real world timeline** correctly. The audio unit system does not implement realtime sample rate (time) modification or timeline adjusting. And if so, the MTC would be generated constantly in correlation to the real time never the less. Otherwise it would not make any sense to produce such kind of absolute, tempo independent real world time code.

In host sync mode there is no possibility to manually adjust the time or beat positions, as everything will be synced to the hosts transport flow.

~~Note: There is another (quite experimental) plugin available, the "MIDI SWEET : Abused MTC". This audio unit allows to modify the generated MTC signal with a variable tempo factor and can be used to correct the resulting time resolution across different sample rate based systems (i.e. between 44.100, 48.000 or 96000 sample rate systems). However Abused MTC is adjustable freely in its virtual tempo.~~

UI Update And Threading

Everything displayed on the user interface is threaded and handled with lower priority. Highest priority is reserved exclusively for the audio thread. We have chosen a threaded timer based approximated display refresh time of the MTC frame rate, which is 24 frames per second. The generated MIDI messages are always guaranteed to be sample accurate and sent with precise timing information in sample rate frequency.

If the UI becomes sluggish, this is probably caused by graphics processing overload and possibly for other reasons by the OS and does not affect the signal generation inside the audio thread in any way.

There may be interruptions on the audio thread too, due to system processing overload, even with mobile devices. Other system interruptions are possible too. In such cases, please reset and rewind your host and the MIDI Time Control, otherwise it may produce false results afterwards.

The unit will stop in bypass mode completely. Neither the graphics nor the generated signals will continue, if bypassed. Bypassing a unit will therefore effectively reset all internal data to zero, at least in receiver mode and with enabled host sync.