Gizmo

An Arduino MIDI Tool, Version 6

By Sean Luke sean@cs.gmu.edu

For Aric

Gizmo is a small MIDI device intended to be inserted between the MIDI OUT of one device and the MIDI IN of one or more other devices. There are a number of tools like this, such as the esteemed MIDIPal[1] or a variety of tools from MIDISolutions.[2]

Gizmo is the unholy union of an Arduino Uno or Arduino Mega 2560,[3] an Adafruit 16x8 I2C LED matrix[4], and a SparkFun MIDI Shield[5] to perform a variety of helpful MIDI tasks. Gizmo has several built-in applications, and assuming there's space[6] you can develop additional applications for it.

This is a software project: the hardware is just three off-the-shelf boards and four wires. There's no enclosure, no custom board. If you'd like to make an enclosure, I'd love to see it.

Gizmo comes built-in with the following applications and other capabilities:

Applications

- **Arpeggiator.** Gizmo's arpeggiator has built-in up, down, up/down, repeated chord, note-assign, and random arpeggios spanning one to three octaves. Additionally, you can define up to ten additional arpeggiator patterns, each up to 32 notes long, involving up to 14 different chord notes, plus rests and (on the Mega) ties. Arpeggios can be *latched* (or not), meaning that they may continue to play even after you have released the keys. You can specify the note value relative to the tempo (ranging from eighth triplets to double whole notes), the note length as a percentage of the note value (how legato or staccato a note is), whether or not the velocity is fixed, the output MIDI channel, and the degree of swing (syncopation). Gizmo will show the arpeggio on-screen.

- **Step Sequencer.** Gizmo's step sequencer can be organized as 12 16-note tracks, 8 24-note tracks, or 6 32-note tracks. Each note in a track can have its own unique pitch and velocity, or be a rest, or continue (lengthen, tie) the previous note. You can also fix the velocity for an entire track, mute tracks, fade their volume, specify their independent output MIDI channels, specify the note value relative to the tempo (again ranging from eighth triplets to double whole notes), the note length as a percentage of the note value (how legato or staccato a note is), and the degree of swing (syncopation). In addition to note data, you can sequence CC, RPN/NRPN, Pitch Bend, Aftertouch, or PC data. The step sequencer lets you edit in two modes: either by triggering independent steps like a classic drum sequencer, or by playing a sequence of notes. The Arduino Uno can store up to two sequences in its slots (shared with the Recorder, discussed next). Gizmo will show and edit sequences on-screen. **Extended Version on the Mega:** The extended version also permits 4 48-note tracks or 3 64-note tracks. You can specify a mute-pattern for each track along four iterations of the sequencer. There's also a "performance mode" which allows sequence chaining, playing along, and transposition, among other things. You can stipulate whether starting/stopping the sequencer will also start/stop the MIDI clock. You can also have the Sequencer avoid playing notes out as they are entered. The Mega can store up to nine sequences.

[1] Though they share nothing in common, MIDIPal is quite similar to Gizmo in functionality and goals. But I wasn't even aware of the existence of MIDIPal when I started building Gizmo. And besides, re-inventing the wheel is fun! I've since ripped off a few of MIDIPal's features. MIDIPal's page is http://mutable-instruments.net/midipal This device has been discontinued, but its open source software lives on in other hardware clones, such as the MIDIGal (https://midisizer.com/midigal/) or MIDIBro (http://www.audiothingies.com/product/midibro/)

[2] http://www.midisolutions.com/

[3] http://arduino.cc

[4] https://www.adafruit.com/categories/326 I use the 1.2-inch 16x8 matrix with red round LEDs, which works well. This particular model can be found at https://www.adafruit.com/products/2037

[5] https://www.sparkfun.com/products/12898

[6] As of this writing, Gizmo on the Arduino Uno uses 32,252 bytes out of 32,256 available. So if you're like to extend Gizmo on the Uno, you'll need to make sure your extension fits within 4 bytes :-) Instead: get a Mega.

- **Recorder.** The Arduino doesn't have a lot of memory, but we provide a note recorder which records and plays back up to 64 notes (pitch and velocity) played over 21 measures. The recorder has 16-voice polyphony. It's enough to record a very short ditty. You can also set the recorder to loop the recording while playing, and to provide a click track. The Arduino Uno can store up to two recordings in its slots (shared with the Step Sequencer). **Extended Version on the Mega:** The extended recorder gives you the option of auto-repeating a recording or stopping when it is finished. The Mega can store up to nine recordings.

- **MIDI Gauge.** The gauge will display all incoming MIDI information on one or all channels. Note on, note off, and polyphonic aftertouch are shown with pitch and velocity (or pressure). Channel aftertouch is shown with the appropriate pitch. Program changes indicate the number. Pitch bends indicate the value in full 14-bit. Control Change, Channel Mode, NRPN, and RPN messages display the number, value, and whether the value is being set, incremented, or decremented. Sysex, song position, song select, tune request, start, continue, stop, and system reset are simply noted. Rapid-fire MIDI signals such as MIDI clock, active sensing, and time code just turn on individual LEDs. **Extended Version on the Mega:** The extended version lets you read raw Control Change messages (that is, not parsed into NRPN etc.), and provides more useful information on Channel Mode messages. The Mega can store up to nine recordings.

- **MIDI Control Surface.** Gizmo has two potentiometers and two buttons. Each can be configured to send a unique Program Change, Control Change, NRPN, or RPN commands. The potentiometers can send up to 10 bits of data (MSB + partial LSB). **Extended Version on the Mega:** The extended version also can send pitch bend, and aftertouch.The extended version also contains an **8-stage loopable envelope** reminiscent of the wave envelope in the Microwave synthesizers, and also a **random LFO**, both of which can manipulate any of these commands as well.

- ***Mega Only:*** **Keyboard Splitter.** You can split incoming MIDI notes by pitch and route them to different MIDI out channels. You can also fade notes between two channels depending on note velocity.

- ***Mega Only*** **MIDI Thru** Gizmo can do magic with incoming notes. It can *merge* and *block* channels. It can also *distribute* notes to different MIDI channels as you play: this lets you play multiple mono synths as if they were polyphonic, for example. Gizmo can also *layer* notes: when you play a note, Gizmo will send some $N > 1$ note-on messages instead of one. This lets you use a polyphonic synthesizer to play a note with more than one voice, making it fatter, but not going all the way to unison. Gizmo can also perform *chord memory*, playing a transposed chord of your choosing starting at a note you are currently playing. Last, Gizmo can *debounce* notes (typically from drum pads), refusing to play a second identical note if it happens to soon after the first one. You can do all of these simultaneously.

- ***Mega Only:*** **Measure Counter.** You can count elapsed beats, measures (bars), and phrases; and alternatively count eighths of a second, seconds, and minutes. The counter also responds to external MIDI clock directives.

- ***Mega Only:*** **Synth MIDI Helpers.** Utilities designed to improve the MIDI of specific synthesizers. At present Gizmo comes with helpers for the Waldorf Blofeld, Korg Microsampler, Oberheim Matrix 1000 (firmware 1.2) and Yamaha TX81Z.

Other Cool Stuff As if this weren't enough, Gizmo also comes with the following capabilities. All of these settings are automatically saved permanently in memory.

- **Tempo and the MIDI Clock.** Gizmo can sync to an external MIDI clock, can ignore it and use its own internal clock, and can emit the same as a MIDI clock. Gizmo can also pass the external MIDI clock through or block it. When using its own internal clock, Gizmo supports tempos ranging from 1 to 999 BPM, settable numerically or via tapping. You can start, pause/un-pause, and stop this clock via

NRPN. **Extended Options on the Mega:** Gizmo can also divide the MIDI clock, thereby outputting a slower clock than it is inputting. And you can also start, stop, and continue clock via buttons on Gizmo or via CC.

Gizmo applications can also be set with a *note value* (or *note speed*) to play their notes relative to this clock, ranging from eighth triplets (1/24 beat) to double whole notes (8 beats). At any time, Gizmo pulses on-screen LEDs indicating the clock and note speed pulses. Finally, Gizmo can be configured to vary the degree of *swing* or *syncopation*.

- **In and Out MIDI Channels** Gizmo can be configured with input and output MIDI channels (some applications will use these: others will have multiple channels and will treat these as defaults). Both channels can be turned off, and the input channel can be OMNI. Gizmo pulses on-board LEDs indicating incoming and outgoing MIDI data.
- **Remote Control via MIDI NRPN.** Gizmo can be controlled remotely via NRPN rather than using its on-board controls. If you start using the controls, Gizmo's onboard potentiometers are locked out so their noise will not interfere with your remote control. You can unlock the potentiometers by pressing a button on-board Gizmo, or by sending Gizmo's *Release* NRPN message. **Extended Options on the Mega:** Gizmo can be remote-controlled via CC.
- **Bypass.** You can quickly put Gizmo in Bypass mode, to pass through all MIDI signals and generate no new ones (with the exception of the Controller, see Section 10). Putting Gizmo in Bypass mode will also send an All Notes Off message, so toggling Bypass is also a useful way to hush your synthesizer.
- **Screen Brightness.** You can change the screen brightness.
- ***Mega Only*** **Transposition and Volume Control.** You can transpose all of Gizmo's MIDI output pitch by anywhere from $-60...60$ steps. Additionally you can multiply the MIDI output velocity (volume) by multiples of two ranging from 1/8 to 8.
- ***Mega Only*** **Menu Delay.** Many of Gizmo's menus display the first few letters of a menu item, then pause for some interval of time before they start scrolling the full item. You can change this interval.
- ***Mega Only*** **Sysex Dumps.** Arpeggios, Step Sequences, and Recordings can be uploaded and downloaded to your computer.

1 Building Gizmo

1. **Assemble the Hardware** Gizmo consists of an Arduino Uno or Arduino Mega, a SparkFun MIDI Shield[7], and an Adafruit 16x8 LED Matrix[8] attached via four wires (I2C). The SparkFun MIDI Shield does not come with headers to plug into the Arduino, and you'll need to get those too: it's assumed you'll either use plain headers, or use stackable headers. I use stackable headers as it makes it easy for me to plug my four I2C wires into it. See Section 1.2 for a warning about the low-quality buttons supplied with the MIDI Shield and what to do about it.

   Attach the four wires for SDA, SCL, 5V (VCC), and GND between the MIDI Shield and the LED Matrix.

2. **Modify your Arduino Software** You will need a pretty recent version of the Arduino software, as newer versions have better compilers which compile more compact code. I developed Gizmo on Arduino 1.6.12[9] for MacOS X. You will want to modify the source code files for the Arduino libraries in two spots:

   Add Nonblocking I2C Writes This will dramatically speed up I2C for our purposes.

   (a) Locate your `Wire.cpp` and `Wire.h` files. On the Mac they're located in `Arduino.app/Contents/Java/hardware/arduino/avr/libraries/Wire/src/`

   (b) Add the following line inside the "public" method region in `Wire.h`:

   ```
   uint8_t endTransmissionNonblocking();
   ```

   (c) Add the following method to `Wire.cpp`:

   ```
   uint8_t TwoWire::endTransmissionNonblocking()
       {
       uint8_t ret = twi_writeTo(txAddress, txBuffer, txBufferLength, 0, 1);
       return ret;
       }
   ```

   Reduce the I2C Buffer Size from 32 to 20 This gives us some extra static RAM space.

   (a) Locate your `Wire.h` file. Again, on the Mac it's located in `Arduino.app/Contents/Java/hardware/arduino/avr/libraries/Wire/src/`

   (b) Change the `BUFFER_LENGTH` constant in the `Wire.h` file as follows:

   ```
   #define BUFFER_LENGTH 20   // Was 32
   ```

   (c) Identify your `twi.h` file. On the Mac it's located in `Arduino.app/Contents/Java/hardware/arduino/avr/libraries/Wire/src/utility/`

   (d) Change the `TWIBUFFER_LENGTH` constant in the `twi.h` file as follows:

   ```
   #define TWI_BUFFER_LENGTH 20    // Was 32
   ```

3. **Download the Gizmo Software** You can grab it with git or svn at `https://github.com/eclab/gizmo/` or download a `.zip` file at `https://github.com/eclab/gizmo/archive/master.zip`

[7] https://www.sparkfun.com/products/12898

[8] This matrix comes in various colors LED shapes (round, square), and matrix sizes (0.8 inch, 1.2 inch). I personally use a 1.2 inch round red matrix. The URL for my model is https://www.adafruit.com/products/2037

[9] Apparently some more recent versions of the Arduino software come with a compiler which doesn't compile as compactly as 1.6.12 did, so you may have problems compiling the standard Uno distribution of Gizmo. If you run into this, try 1.6.12, I know that works for sure.

4. **Install the FortySevenEffects MIDI Library** But not straight off of its website.[10] The MIDI library is in a state of flux and is having major additions being made. The Gizmo code is extremely sensitive to code size (at least on the Uno). Thus for now, use the version I include in Gizmo's Github repository.

5. **Install the SoftReset Library** A copy of it is in Gizmo's Github repository.

6. **Switch the MIDI Shield to "Prog"**

7. **Build and Upload Gizmo's Code to the Arduino** It should compile cleanly for either an Arduino Uno or an Arduino Mega 2560. On the Uno it'll complain of Low Memory.

8. **Switch the MIDI Shield to "Run"** You can plug in MIDI cables now.

9. Do a Full Reset of Gizmo Power up the Arduino. Then hold down all three pushbuttons on the MIDI Shield. While doing so, press and release the reset button on the MIDI Shield. Continue to hold down the three push buttons until the two LEDs on the MIDI Shield light up.[11] Let go of the pushbuttons.

   This procedure initializes the memory, options, and files in the device. Now you're ready to go!

1.1 Customizing Your Arduino

Gizmo is fairly modular: you can choose which applications you want to have on your device, as well as various features. This allows us to (for example) have more applications and features on the Mega than on the Uno. If you want to port to a device with a somewhat tighter memory footprint (such as the Arduino Mini), or want to make room on the Uno for your own application, you just need to change which options are installed. See the file `All.h`

1.2 Buttons, Shield Quality, and Debouncing

The SparkFun MIDI Shield uses poor quality buttons. You can push down on the button and wiggle it and it'll think it's been pressed multiple times. As a result occasionally pressing a button may send two or more button-presses to Gizmo.

Gizmo has a debounce mechanism built-in: it ignores secondary apparent button-presses if they occurred within N milliseconds of an earlier button release. At present N is set to 100ms. If you need to press buttons faster than ten times a second (and it's possible) set the `BUTTON_PRESSED_COUNTDOWN_DEBOUNCE` constant (in `TopLevel.h`) to something smaller.

After about four months of extremely heavy use, my select and back buttons began to fail: if I pressed either one the right way it wouldn't indicate that it's been pressed. If you reach this state you may have to replace the buttons, though that's kind of hard to do: I just built a new board with different, better-quality low-profile buttons (just look on Adafruit for 12mm pushbutton switches). Fingers crossed.

[10] https://github.com/FortySevenEffects/arduino_midi_library

[11] Sometimes you have to do this a couple of times on the Mega to get the lights to light up.

2 Initializing Gizmo

Gizmo must be initialized before it can be used. You can do a factory reset on Gizmo with this procedure:

1. Hold down all three buttons
2. Press the reset button
3. Continue to hold down all three buttons until both on-board LEDs have lit.
4. Let go of the buttons.

Gizmo will reset all of its flash RAM and reboot. This means that all options will be set to their default state, and all file storage (including arpeggios) will be emptied.

Doing a "Partial Reset" Whenever you load new Gizmo software, you should reinitialize Gizmo or strange things will happen. However a full factory reset also erases your saved files (sequences, arpeggios, recordings). If you want to preserve these but just reset the options, just hold down the left and right buttons instead of all three (don't hold down the middle button).

2.1 Starting Gizmo Headless

Gizmo normally targets the SparkFun MIDI Shield, but you might be able to use its code with some other MIDI shield, as long as the shield just intercepts Serial to do its MIDI and doesn't interfere with I2C. Such MIDI shields are unlikely to have the button, pots, or LEDs that the SparkFun shield has. But that's okay, you can control the buttons and pots virtually via NRPN, and just do without the two LEDs.

To do this you'll need to clear Gizmo's options and files without needing to press any buttons (as you won't have any). You can do it like this:

1. In the `All.h` file, change the line

   ```
   //#define HEADLESS_RESET
   ```

   ...to...

   ```
   #define HEADLESS_RESET
   ```

2. Compile and download the code, and run it once. Gizmo will show its opening screen, then display `RSET` and not do anything else.
3. In the `All.h` file, change the line back, that is, to

   ```
   //#define HEADLESS_RESET
   ```

Gizmo's options have now been reset in a slightly different way than usual: specifically, the **Control MIDI** option (see "Control MIDI" in Section 16) has been set to MIDI channel 16, rather than to OFF.

Now you need to modify Gizmo's code to run headless, so it doesn't bother lighting up the on-board LEDs or reading the buttons and pots. You can do this as follows:

1. In the `All.h` file, change the line

   ```
   //#define HEADLESS
   ```

   ...to...

```
#define HEADLESS
```

2. Compile and download the code, and run it.
3. You should now be able to control Gizmo via NRPN (see "Control MIDI" in Section 16) on channel 16.
4. In the `All.h` file, you might wish to change the line back, that is, to

```
//#define HEADLESS
```

I make no guarantees that this will work: but it might.

3 Starting Gizmo

If you power up Gizmo, or press its reset button, it will display its **boot screen**, which includes the version number (at right, the version number is "1").

Gizmo will then show the **root menu**. Use the left potentiometer to scroll to any of the following applications, and use the select button to select an application. Available applications are:

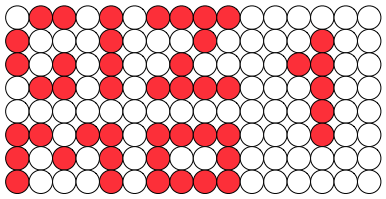


Figure 1: Boot Screen

3.1 A Bootloader Bug on the Arduino Mega

Many versions of the Arduino Mega have an error in their bootloader which may cause the Arduino to hang if you've got a MIDI device cables plugged into MIDI IN when you power Gizmo up. Similarly, if you press the reset button, it'll look like it's not booting up, but just frozen. The issue is simple: if on bootup the Mega hears data coming in on MIDI IN (which is its serial port), the bootloader assumes that you're downloading a new program and immediately goes into program-load mode. But in fact it's probably just MIDI Clock or Active Sensing commands from a chatty synthesizer. You have four options:

- Turn on or Reboot Gizmo without anything plugged into MIDI IN, then plug the cable in.
- Turn off the synthesizer, turn on or Reboot Gizmo, then turn the synthesizer back on.
- Turn on or Reboot Gizmo with its switch set to "Prog", then when it's running, switch it to "Run". I suggest this one.
- Permanently fix the issue by burning a new bootloader. This is an involved process, and I don't suggest it. See https://github.com/arduino/Arduino/issues/2101

4 User Interface

Gizmo has a modest interface: a 16x8 LED display, two additional on-board LEDs (red and green), three buttons, and two potentiometers (dials): but it strives hard to make good use of these.

Gizmo's top-level interface layout is a menu hierarchy. You select menu items, which may take you deeper into the hierarchy, or you can go back up towards the top of the hierarchy. The top level menu chooses between different **applications**. Once you have entered an application, its **application number** is displayed in the **current application region** (see Figure 2).

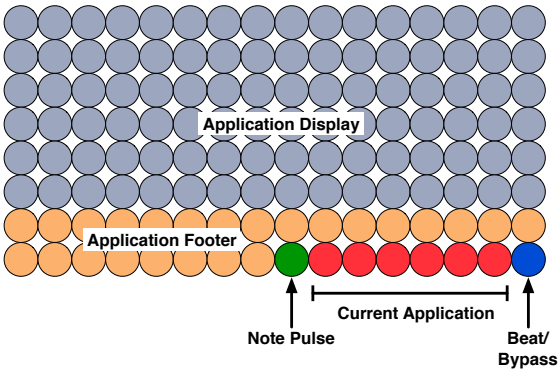


Figure 2: High-Level Gizmo Layout

The current application LEDs light up right-to-left in a certain pattern with increasing application numbers, as shown in Figure 3. Gizmo's interface can presently support up to eleven top-level applications.

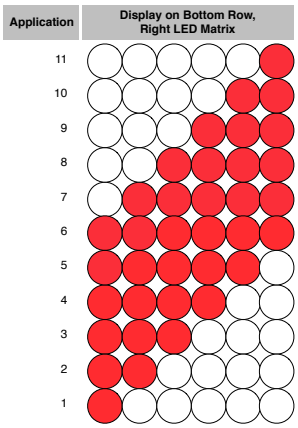


Figure 3: Application Values

Also shown in Figure 2 are the **Beat/Bypass** light and the **Note Pulse** light. The Beat/Bypass turns on and off every *beat* (quarter note). This shows the current **tempo**.[12] The Note Pulse light turns on and off every *note pulse*: this is the speed you have chosen for the arpeggiator, step sequencer, etc. Note pulses are defined in terms of note value: for example, if the note pulse is presently in sixteenth notes, then this light will turn on and off four times faster than the beat light, which is in quarter notes. The note pulse light is also affected by the degree of swing (syncopation) you have defined.

If Gizmo's clock is presently **stopped**, either because you have stopped it internally or because Gizmo is being driven by an external MIDI clock which is currently stopped, then **both the Beat/Bypass and the Note Pulse lights will be off** (though if Bypass mode is on, then Beat/Bypass will still blink rapidly). See Section 17.2 for more information about starting and stopping the clock.

The reset of the screen is given over to the application. In many cases an application will display text and other data in the **Application Display** region, and light certain status LEDs in the **Application Footer**.

You wend your way through menus, choose applications, and manipulate them using Gizmo's three buttons and two potentiometers as follows:

Buttons The left button is called the **back button**. The right button is called the **select button**. The center button is called (unsurprisingly) the **middle button**. The buttons respond to being **pressed** and immediately released, and also respond to **long presses**: holding a button down for a long time (presently one-half second) and then releasing it.

Buttons do different things. Here are certain common items:

- **Pressing the Back Button** This is the "escape" or "cancel" button: it **always goes back up the menu hierarchy.** Thus you can't ever get lost — just keep pressing the Back button and you'll eventually find yourself at the root menu.
- **Long-Pressing the Back Button** This toggles **bypass mode**, where Gizmo tries hard to act as if it's just a MIDI THRU coupler between its input and output MIDI wires: it passes through all the MIDI it receives, and with one exception (the Controller, see Section 10) it doesn't output any new data. In bypass mode, the **Beat/Bypass** light (Figure 2) will stop beating and instead will flash rapidly. Bypass

[12] A beat occurs every 24 *pulses* of a MIDI Clock or Gizmo's internal clock.

mode also sends an "all sounds off" command to all outgoing MIDI channels, so if you need to shut up a misbehaving synth, you can enter bypass mode and then leave it again.[13]

- **Pressing the Select Button** If you're being presented with a menu of options, or a number to choose, or a glyph to choose, the Select button will select the currently-displayed option. In some applications (such as the Step Sequencer, Arpeggiator, and Recorder) the Select Button is assigned other tasks.
- **Pressing the Middle Button** Likewise if you're being presented with a menu of options, or a number to choose, or a glyph to choose, the Middle button will increment the current display (but not select it).

Potentiometers Gizmo's potentiometers (or **pots**) do a number of tasks, including scrolling through menu options, moving a cursor horizontally or vertically, or choosing a number. The left pot does primary tasks, and the right pot assists when appropriate. Note that these are potentiometers, not encoders: if you start turning one, you may experience a big jump initially. The potentiometers have a limited range (0...1023), and a significant degree of noise, so realistically they can only choose numbers between 0...127 or so. If asked to select a number from a *big* range, the **left pot** will act as a coarse-level selector, and the **right pot** will fine-tune the number.

On-Board LEDs There are two LEDs on-board the SparkFun MIDI Shield. The **red LED** is toggled on and off to reflect incoming MIDI data. The **green LED** is toggled on and off to reflect outgoing MIDI data.

[13] By default Gizmo starts up with Bypass turned on in a special way: when you select your very first application, Gizmo will automatically turn Bypass off (without sending an "all sounds off"), enter that application, and thereafter treat Bypass normally. This only happens on selecting your first application, and that's it. Also if you toggle Bypass manually, Gizmo will automatically start treating Bypass normally. The point of this is to allow Gizmo to be in Bypass mode unintrusively as soon as it's powered up so you can have it attached to your keyboard all the time even if you don't use it that often. If you would prefer that Gizmo start up with Bypass turned *off*, in `All.h` change the line `#define TOPLEVEL_BYPASS` to `// #define TOPLEVEL_BYPASS` and rebuild Gizmo.

5 How Gizmo Handles MIDI

Gizmo listens for certain kinds of incoming MIDI data, then hands it to an **application**, which may in turn send MIDI data out. Applications can send data out any way they wish, but in many cases they restrict themselves to sending data out only to Gizmo's **default MIDI Out Channel.** Note data (Note on, Note Off) sent out may also be subject to further user-specified transformation in the form of transposition or volume control.

Channel Data If the incoming data is *channel data* (Note on/off, Aftertouch, Pitch Bend, Program Change, Control Change / NRPN / RPN), most applications will listen to it only if it is from the **default MIDI In Channel**, which can be 1–16, or OMNI (all channels), or OFF (no channels). Some applications may listen to channel data from certain additional channels: for example the MIDI Thru Facility also listens to data on an additional "Merge channel". **Data from all remaining channels is simply passed through to MIDI Out.**

Clock Data Gizmo can follow clock data or ignore it and generate its own clock. Furthermore Gizmo can pass incoming clock data through to MIDI Out, or block it. Gizmo can also emit its own clock data.

Other Data Most other data is simply passed through from MIDI In to MIDI Out.

Bypass Mode When bypass mode is engaged, all data received is simply passed to MIDI Out. Applications may still listen to

Controlling Gizmo One incoming channel can be defined as the **default MIDI Control Channel**. Data on this channel is used only to manipulate Gizmo (via NRPN or CC), for example, changing a parameter.

6 The Arpeggiator

The arpeggiator allows you to play different kinds of arpeggios, and also to create and save up to ten of them. The arpeggiator's main menu lets you choose to play five different kinds of pre-set arpeggios, or to select an arpeggio from user-created arpeggio slots 1–10, or finally to create an arpeggio.

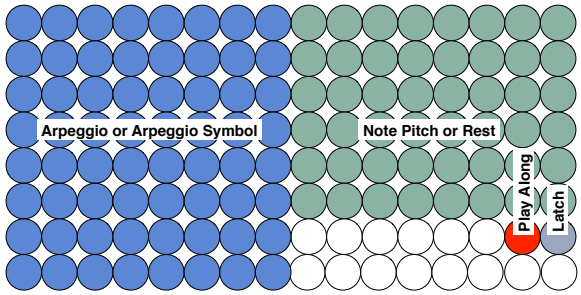


Figure 4: The Arpeggiator Display

An arpeggio is a pattern for repeatedly playing the notes in a chord. When you hold down a chord on the keyboard, the arpeggiator uses this pattern to play the notes, typically one at a time, according to the pattern selected. Patterns do not specify exact notes, but rather note orderings. For example, an arpeggio might be defined as 1, 2, 2, 3, 4, 2, 3: if 1 is set to be the root (lowest) note of your chord, then if you play a C7 chord (C E G B♭), then the arpeggiator will repeat the sequence C E E G B♭ E G. This pattern will repeat as long as you hold down the keys in the chord.

Alternatively if you toggle the **latch**, then the playing will continue after you have released the keys, and will only shift to a new chord when you release all notes and then hold down an entirely new chord. Latch mode is toggled by Long-Pressing the Middle button. When Latch mode is engaged, an LED will light as shown in Figure 4. Latch mode is stored permanently in memory, so next time you start up the Arpeggiator, your option will be remembered.

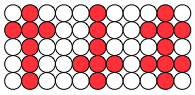


Figure 5: Up, Down, and Up-Down Symbols

(Mega Only) The arpeggiator also has a **play-along mode** which you toggle by pressing the Middle button. In this mode, if you press notes on the keyboard, they are not sent to the arpeggiator, but instead are routed out a MIDI channel of your choosing. This is useful in three contexts. First, you can use it as a lightweight bypass mode of sorts. Second, you can use it to suddenly start playing a different synthesizer than the one on which you're doing arpeggios. Third, and most importantly, you can use this in combination with latch mode: enter latch mode, get an arpeggio going, and then switch to play-along mode to use the keyboard for other purposes (playing along with the arpeggio for example).

Playing Preset Arpeggios There are six preset arpeggios: *up*, *down*, *up-down*, *random*, *chord repeat*, and *assign*. They work as follows:

- **Up**. Each note in your chord is played in turn, lowest note first. When all notes are exhausted, the pattern is repeated.
- **Down**. Each note in your chord is played in turn, highest note first. When all notes are exhausted, the pattern is repeated.
- **Up-Down**. Each note in your chord is played in turn, lowest note first, up to but not including the highest note. When all notes are exhausted, all notes in the chord are played in turn, highest note first, down to but not including the lowest note. When all notes are again exhausted, this pattern repeats.
- **Random**. Chord notes are played at random. Gizmo tries to not play the same note twice in a row, unless there's only one or two notes in the chord.
- **Chord Repeat**. The entire chord is repeatedly played in its entirety.
- **Assign**. Each note in your chord is played in turn, in the order in which you had pressed the keys when you formed the chord. When all notes are exhausted, the pattern is repeated.

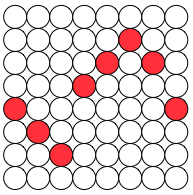
As you play a preset arpeggio, the notes played by the arpeggio will appear at right. The symbol representing the preset will appear at left.

While playing a preset arpeggio, you can select the number of octaves by calling up the play menu (Long-Pressing the Select button), then choosing **Octaves**. If you have an octaves value ≥ 0, then when the arpeggiator exhausts its notes, it will repeat them again one octave higher, then another octave higher, and so on, up to the number of octaves chosen. You an select any value from 0–6 octaves.[14]

By default the key velocity[15] that *all* notes in the track will use (as a value 0...127) is determined by the **first note** you press when you hold down a chord. This strategy usually works well, but if you can instead fix the velocity to a specific value by (once again) calling up the menu by Long-Pressing the Select button, then choosing **Velocity**. At the far highest end of the velocity options is `FREE`, which means that the velocity is determined by your fingers again (this is the default).

Creating an Arpeggio The last menu option in the top-level Arpeggiator menu is **Create**. If you choose this, you can create an Arpeggio. To do this you enter notes one by one. These won't be the *actual* notes that are played when you hold down a chord. Rather, after you save the result, the Arpeggiator will compact these notes into a sequence of consecutive numbers. For example, if you play C3 E3 G3 E3 G2, then the Arpeggiator will compact these into 2 3 4 3 1. Then when you play this as an arpeggio, the Arpeggiator will assign each of these numbers, in order, to the notes in your chord.

The first question you will be asked is `NOTE`: the Arpeggiator is asking you to specify the note in your arpeggio pattern which will correspond to the root (lowest) note in a played chord. This allows you to make a pattern which plays notes both above and below the played chord. Consider the above pattern (C3 E3 G3 E3 G2, thus 2 3 4 3 1). If you assigned C3 (thus 2) to be the root, then the Arpeggiator will play its final note (1) *below* the lowest not in your played chord.



Example Arpeggio

After you have entered the root, the Arpeggiator will wait for you to start playing notes. You can enter a sequence of up to 32 notes, including 15 unique notes (14 on the Mega). If you enter to many unique notes, or more than 32 notes in total, the Arpeggiator will simply refuse to accept the next note. You can also enter rests by pressing the Middle button. On the Mega you can also enter ties (stretching the last note) by long-pressing the Middle button. The first note may not be a tie: what would it be tying?

As you enter notes, they are displayed in the left half of the LED, and the latest note pitch is displayed in the right half. Figure 6 shows different arpeggio patterns. If you have eight or fewer unique notes in your pattern, then they will be displayed in *sparse mode* as shown at right. If you have over eight notes, they will be displayed in *dense mode* in order to show all of the notes. The LEDs don't show the notes per se, but rather the consecutive numbers that the notes represent. For example, the ascending notes shown at right could be C, D, E, F, G, A, B, C, or they could be C, C♯, D, D♯, E, F, F♯, B.

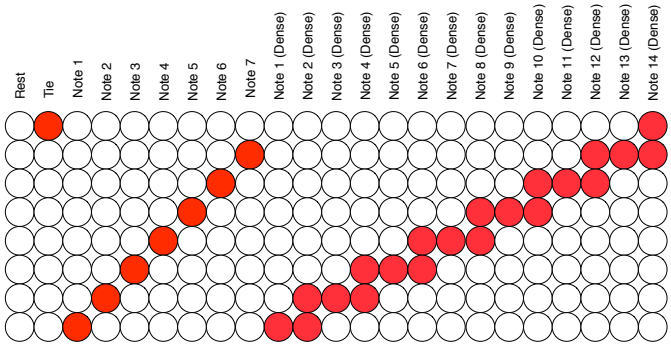


Figure 6: Displayed arpeggio notes. If you have entered more than 7 unique notes, the arpeggio will switch to "dense mode" to display them.

If you don't like the latest portion of your arpeggio, you can scroll the cursor back to an earlier section by turning the right knob. Then if you start playing, the later notes will be discarded and the arpeggio pattern will continue from where the cursor was. You can also scroll to view your past notes, then scroll all the way back to continue.

[14]Note that this means that if your chord is larger than one octave, you may not get what you expected. For example, if you are playing UP with C5 E5 G5 D6, and with octaves=1, then the arpeggiator will play C5 E5 G5 D6 C6 E6 G6 D7. Note that C6 is played after D6 is played. That's the current behavior right now anyway.

[15]By velocity we mean in the MIDI sense: how fast the key is struck; this basically translates to loudness.

An **important note regarding the scrolling.** The cursor is always located at the position where new notes will be entered. But as you scroll back and forth, you'll hear the notes of your arpeggio. When you scroll and hear a note, that note is the note *immediately before* the cursor position. It's the last note in the arpeggio before you start adding new notes. It is *not* the note that you'll replace when you start entering notes.

Anyway, when you are satisfied with your arpeggio, Long-Press the Select button to save. You'll be asked for a slot to save in, and after that, you'll be back in the main Arpeggiator menu.

Playing a User-Created Arpeggio To play a user-created arpeggio, from the main Arpeggiator menu, choose one of the numbers 0–9, each of which represents the arpeggiator slot you had saved your arpeggio in (these options appear after the preset options). Then hold down a chord! As you play a user-created arpeggio, the notes played by the arpeggio will appear at right. The arpeggio pattern will scroll by at left.

The preset arpeggio patterns (such as Up-Down) vary with the number of notes in your chord: if you hold down five notes for example, and octaves=0, then five notes will be played in the arpeggio pattern. This isn't the case for user-created arpeggios, which have a fixed user-specified number of notes.

What happens if your arpeggio is designed for three notes but you hold down five notes in your chord? Only three of your five notes will be played.

What happens if your arpeggio is designed for five notes but you hold down only three notes in your chord? Then notes from the bottom (or top, as appropriate) of your chord will be transposed up/down one octave to make up the remaining notes.

Also, because they are fixed in size, user-created arpeggios do not respond to the octaves option.

(Mega Only) If you're currently playing a user-created arpeggio, and you press the Select button, you will immediately **advance to the next non-empty user-created arpeggio**. You might use this to shift between a sequence of arpeggios of interest.

More Arpeggiator Options In addition to Octaves, the arpeggiator menu (Long-Press the select button) also includes the Options Menu as a submenu.[16] See Section 16 for detailed information on each of the options. The Arpeggiator responds to the following options:

Tempo	Note Speed	Swing	Note Length	Transpose (on Mega)
In MIDI	Out MIDI	Control MIDI	MIDI Clock	Volume (on Mega)

Of particular interest to you may be Length and Tempo. And Swing!

One note regarding the **Note Length** option. If the Note Length is set to 100 (and *only* 100), the Arpeggiator behaves specially: each note will extend until the next note (or rest) is played, regardless of how long that is. Thus if you have Swing turned on, notes will extend until the next syncopated (delayed) note even if their time is up. This provides a fully legato feel.

The Left and Right Knobs When playing an arpeggio, turning the left knob will transport you directly to the Tempo menu option, so you can easily change the note tempo. Similarly, turning the right knob will send you to the Note Length menu option. You'll have to turn a lot to get this to happen (as a safety precaution).[17] Once you're done changing the tempo or note length, press the Select button to save it, or the Back button to cancel and return to your previous value.

[16] Recall that the Options Menu is for options shared among several applications: that's why it's a separate menu. That and code space issues.

[17] *What about Arpeggio Key Velocity? Swing? Note Speed? Velocity or Transpose?* Arpeggio Key Velocity's easy: just press the select button twice. For the others you'll have to go through the options menu, sorry. Though it's easy to modify Gizmo to change this.

7 The Step Sequencer

The Step Sequencer makes it easy to create multitrack loops common on devices like drum machines. On the Uno, you can have twelve 16-note tracks, eight 24-note tracks, or six 32-note tracks. On the Mega you can additionally have four 48-note tracks, three 64-note tracks.

When you choose the Step Sequencer from the root menu, you will be asked to load a sequence from a slot. You can do this, or you can create a new empty sequence by choosing - - - - (meaning *none*). Slots are displayed by number, plus an L (for Load) and an optional R (for Recorder) or S (for Step Sequencer) if the slot already has a file from one of the two applications. This letter **blinks** if the file isn't the type of your current application to warn you that you might eventually overwrite a slot when you save.

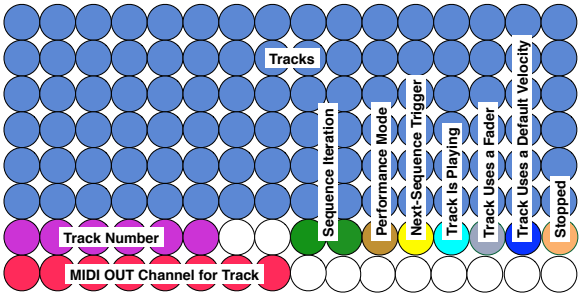


Figure 7: Step Sequencer Display

If you have chosen *none*, or have chosen an empty slot or one presently filled by the Recorder, you'll be asked to initialize the sequence first, choosing from 16, 24, 32, or (on the Mega) 48 or 64 notes per track. Then you are taken to the main Step Sequencer display.

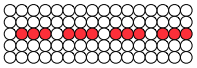


Figure 8: None

Organization Figure 7 shows the general step sequencer display structure. The *Tracks* region is where sequencer tracks are displayed and edited. The *Track Number* area tells you the number of the currently-edited track. Figure 9 explains how to interpret this. The *Stopped* LED tells you if the step sequencer is currently stopped (or playing). The *Track Uses a Default Velocity* LED tell you if all notes in the current track are fixed to the same velocity, or permitted to use different velocities. The *Track Uses a Fader* LED is lit when the pre-track fader (volume) is set to something other than maximum volume for the current track. Finally, the *MIDI Out Channel for Track* tells you what channel the current track is playing. The patterns for the MIDI Out Channel for Track are the same as those in Figure 25, except that MIDI Out does not support ALL (OMNI). Other LEDs are Mega-Only and discussed later.

Pressing the Select button will toggle whether the Step Sequencer is playing or stopped. When you start playing the Step Sequencer, it waits for the clock to reach a measure boundary before beginning. Note that if you start playing the Step Sequencer, but the clock isn't running, it won't play at all. You'll need a clock (internal or external) to drive it.

You'll enter notes (or other kinds of sequenced data) in one of several *tracks*. You select your current track by turning the left knob. In a given track, you choose the current note position by turning the right knob. When the Step Sequencer is playing, the tracks region will also display the *play position cursor*, a blinking vertical set of LEDs which indicate which notes in the tracks are currently playing. This vertical group heads to the right, eventually wrapping around.

Each track takes up one or two rows in the Tracks area: 16-note tracks take a single row, 24-note and 32-note tracks take two rows, 48-note tracks take three rows, and 64-note tracks take four rows. Depending on the length of each track, you could have up to twelve tracks: but the tracks display region is only six LEDs high. This of course means that the Tracks region scrolls as you turn the left knob.[18]

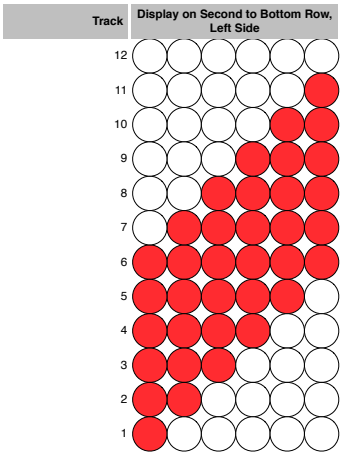


Figure 9: Track Numbers

[18]This also means that can only see one 64-note track at a time. BTW, you might be interested in knowing why Gizmo can't presently do 8- or 12-note tracks. This is because their memory usage is too high — we're storing a lot of per-track information, and with 8-note tracks you can have 24 of them! There's a different reason we don't have 128-note tracks: one track would take up the entire screen, including any footer information. Also, you'd only have a single track, which seems to me to be not particularly useful.

Each *iteration* of the step sequencer (16, 24, 32, etc. notes) it may or may not play a given track depending on whether the track has been **muted**, **soloed**, or what **play pattern** you have specified for it. In the third case, you can stipulate which of every four iterations a track should or should not play. For example, if your sequence has sixteen steps, then you can create a pattern which plays a given track for the first sixteen steps, then doesn't play it for the next sixteen steps, then plays it again for the third and fourth sixteen steps. This is particularly useful for playing a certain beat much less often than others. The current iteration is shown in the Step Sequencer with a two-LED design, as shown in Figure 10, and the *Track is Playing* button lights if the track is supposed to be playing in the current iteration.

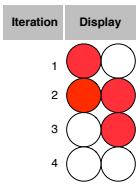


Figure 10: Four Sequence Iterations.

Modes The Step Sequencer can be in one of three modes: the **Edit Mode**, the **Play Position Mode**, and the **Performance Mode**. In the Edit mode you can enter notes, rests, or ties one by one directly in positions of your choosing. In Play Position Mode, when you play notes, they are entered right where the play position cursor is located. This is good for doing on-the-fly recording.[19] Finally in Performance Mode, when you play notes, they're not entered into the Step Sequencer: instead, they're routed to MIDI Out so you can play along with your recorded sequence; or they're interpreted as transposing the sequence as it plays. Performance Mode also allows you to chain sequences and do other nifty things.

Edit Mode Typically you'll start in Edit Mode. Somewhere on the screen there will be a single-pixel blinking *edit cursor* which determines where notes are entered if you play them. The location of this cursor (track, note position) is determined by the dials.

You enter notes by moving the edit cursor to the a location, then playing keys. You can also press the Middle Button to enter a rest, or Long Press the Middle Button to enter a *tie*. A tie just means "keep playing the previous note": it essentially lengthens the previous note. You can enter multiple ties in series to make a note as long as you like.

A track can store and sequence other kinds of data besides Note-On and Note-Off. Specifically, you can set a given track to store and replay any one of (raw) CC, 14 Bit CC, NRPN, RPN, Pitch Bend, Channel Aftertouch, or Program Change commands by choosing **Type** from the menu (see **Menu Options** below). If you select (raw) CC, 14 Bit CC, NRPN, or RPN, you will be further prompted to specify which parameter number you'd like to sequence (for example, CC# 42 or NRPN# 12312).

Non-note data is entered differently than note data. To enter non-note events, send an event to the MIDI In Channel, then press the **middle button**: the most recent sent event will then be recorded at the current position. You can erase data at the current position (essentially send a rest) by long-pressing the middle button.[20]

Some notes. First, if you sent a Polyphonic Aftertouch event (on any key) it will be recorded instead as Channel Aftertouch. Second, if your track is configured for (raw) CC, then you can send *any* raw CC value to it (all 127 parameter numbers), and it will also emit the same; but if your track is configured for 14-bit CC, then only the first 31 parameter numbers may be used. Third, any Pitch Bend value can be stored except for 8191 (the high value). Fourth, any 14 Bit CC, RPN, or NRPN value can be stored except for 16383 (again, the high value).

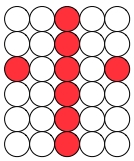


Figure 11: The Play Position Mode Cursor. The current track is marked by the two dots.

Play Position Mode If you turn the right knob fully to the left, you will find that the cursor scrolls right off the left side of the screen. At this point you are in **play position mode**. Here, if you press a key, it is entered at the play position cursor.

In play position mode, the function of the Middle Button changes. Now if you press the Middle Button, it will toggle muting the current track. You can tell a track is muted because the LEDs at each end of the

[19] I myself spend most of my time in Play Position Mode

[20] This may be confusing as rests are entered by simply *pressing* the middle button for note data, but I figured entering data was more important than entering rests in the non-note case. Sorry.

track are blinking. Additionally, if you Long Press the Middle Button, it will erase all the notes on the current track (but not change its settings).

You can tell you're in play position mode because the play position cursor has changed to include two dots which indicate the current track, as shown in Figure 11. When stopped, the play position cursor will sit at the far right of the screen (so you'll only see one dot).

You can get out of play position mode just by turning the right knob so that the edit cursor is once again on-screen.

Performance Mode (Mega Only) If you simultaneously Long-Press the Select and Middle buttons when in either Edit or Play Position Mode, you'll enter Performance Mode. This mode is meant to support performing with completed sequences. Here, if you play notes or CC values different things might happen, depending on how you've configured things: either the notes will be entered on the track (which will be cleared first if an iteration just transpired or the user just switched tracks), or the notes will be routed to some channel in MIDI Out, or the notes will be used to *transpose* the sequence as it's playing. A track is transposed only if you have turned its "Transpose" option on (see Menu Options below).

In Performance Mode, a number of controls change their function Pressing the Back Button doesn't exit the Step Sequencer: but rather, it exits Performance Mode back into whatever previous Mode you had used. Pressing the Middle Button toggles muting of the track. Long-Pressing the Middle Button toggles Soloing: only your current track is played. You can tell you're in Solo mode because every track *except* for the soloed one appears to be muted (LEDs at each end of the tracks are blinking). Pressing the Select button starts and stops playing as usual, and long-pressing the Select button brings up the menu. The Left Knob changes tracks as usual, but the Right Knob changes the **tempo**.

In Performance Mode, sequences can be **chained together**. You can specify that a given sequence will **repeat** for some number of iterations, and then will either stop or will move on to *another sequence*. Since the Mega can hold up to nine sequences, you can use repeating and chaining to make some fairly long songs. Per-track patterns, repeating, and specifying the next sequence are done in the Menu Options (see later).

You can optionally chain two sequences together but set the first sequence to repeat *forever*: how do you ever get to the next sequence? You do it manually: long-press the Select and Middle buttons at the same time, and you will trigger the Sequencer to switch to the next sequence at the end of the current iteration.

In Performance Mode, as in Play Position Mode, you'll always see the Play Position cursor. When in Play Position Mode, a particular LED will also constantly blink to indicate the mode. The current sequence iteration is also shown with a two-note pattern as shown at right. Finally, if you have triggered the Sequencer to switch to a new sequence, then an LED is displayed to indicate this as well. See Figure 7.

The behavior of **Mute** and **Solo** change in Performance Mode. The purpose of these behaviors is to make mute and solo more useful for syncing up with you in performance.

If you press solo, you won't *toggle* the solo. Instead, you *schedule* the solo to be toggled at the beginning of the next sequence iteration. If you press a second time, you undo your previous action. So in short, pressing buttons go through this loop, which is known as **advancing solo**:

- If you are currently *not* soloed: **Schedule Solo** → **Undo**
- If you are currently soloed: **Schedule Un-solo** → **Undo**

Similarly, if you press mute, you *schedule* the track to be muted (or not), at the next sequence iteration, for exactly one iteration, and then to return to its present state. If you press mute a second time, you schedule the track to be muted (or not) permanently at the next sequence iteration. If you press mute a third time, you undo all this. This is known as **advancing mute**:

- If current track is *not* muted: **Schedule Mute For One Iteration** → **Schedule Mute** → **Undo**
- If current track is muted: **Schedule Un-mute For One Iteration** → **Schedule Un-mute** → **Undo**

If you include a CC control surface (see *Remote Control over CC* below) you have the further option of advancing mute on any track individually. This is useful to perform with Gizmo as a groove surface, where you trigger various tracks to play once or turn them on or off during the performance.

Menu Options If you Long Press the Select button, you'll call up the Step Sequencer's menu. The Step Sequencer has a number of options:

- **Solo** (or **No Solo**) This plays only the current track, muting the others.
- **Reset Track** This wipes out and resets the track. Unlike Long-Pressing the Middle Button while in Play Position mode, this doesn't just clear the track: it also resets all of its settings to their defaults.
- **Length (Track)** This specifies the note length of notes in the track, as a percentage (0 is fully staccato and 100 is fully legato). You can choose numerical values, or choose `DFLT`, which tells the Step Sequencer to use the default note length specified in the Options menu (see Section 16).
- **Out MIDI (Track)** This specifies the MIDI Out Channel for notes in the track, You can choose a value 1...16, or choose `DFLT`, which tells the Step Sequencer to use the default MIDI Out Channel specified in the Options menu (see Section 16).
- **Velocity (Track)** This lets you specify a key velocity[21] that *all* notes in the track will use (as a value 1...127). If you would instead prefer that each note in the track use its own independent key velocity as you had entered them, you can choose *none*.
- **Fader (Track)** This lets you specify a volume modifier which is multiplied against the velocities of the notes in the track. Possible values are 0...31, where 0 completely hushes the notes, 16 lets them play at their specified velocities, and 31 is roughly twice their velocity. 8 is one half-velocity, 4 is one quarter velocity, and so on. In general, values larger than or smaller than 16 will make the notes louder or quieter respectively.
- **Type (Track)** (Mega only) Lets you specify the type of data stored in this track. See discussion of **Edit Mode** above.
- **Pattern (Track)** (Mega only) Lets you specify the four-iteration pattern for this track. An `O` indicates that the track will be played, and a `-` indicates that it will be hushed. Your options are: `OOOO`, `O-O-`, `-O-O`, `OOO-`, `---O`, `O--O`, `-OO-`, `OO--`, `--OO`, `OO-O`, `--O-`, `R1/8`, `R1/4`, `R1/2`, `R3/4`, or `EXCL`. The four which start with `R` hush at random: the fraction indicates the probability that the track will be played. `EXCL` is *exclusive random:* if multiple tracks are designated as `EXCL` then at any time exactly one of them will be played (and the others will be hushed). If only one track is `EXCL`, it's always played. Note that many of the non-random patterns are in pairs so you can swap the track being played.
- **Edit** (Mega only) A submenu for some simple cut/copy/paste-like operations:
  - **Mark** Records the current track and edit position as the *mark position*.
  - **Copy** Copies the data at the mark position to the current track starting at the current edit position. The copied data is wrapped around when it goes beyond the end of the current track.
  - **Splat** Copies the data at the mark position to the current track starting at the current edit position. The copied data is *not* wrapped around when it goes beyond the end of the current track: it just stops there.
  - **Move** Works like Copy, except that the mark position track is then erased.
  - **Duplicate** Duplicate absolutely all of the data in the mark track to the current track (including note data, volume, MIDI out, etc.). The specific mark (note) position is ignored.
- **Transpose (Track)** (or **No Transpose (Track)**) (Mega only) This toggles whether this track responds to the user transposing the sequencer during performance mode.

[21] By velocity we mean in the MIDI sense: how fast the key is struck; this basically translates to loudness.

- **Clock Control** (or **No Clock Control**) (Mega only) This toggles *clock control mode*. Normally, if you play or stop the sequencer, it will not send MIDI STOP or MIDI PLAY commands — if you want to start the MIDI clock you have to do so separately (see Section 17.2). However it's common for sequencers to issue these commands when you start or stop the sequencer, so as to also start or stop other sequencers listening over MIDI. To do this, turn on clock control mode. Note that this only has an effect if Gizmo is **generating** a MIDI Clock (see "MIDI Clock" in Section 16). Finally note that regardless of the setting of this options, if Gizmo is *using* an external MIDI Clock (again, see "MIDI Clock" in Section 16), the Step Sequencer will stop and start in response to incoming MIDI Clock commands.

- **No Echo** (or **Echo**) (Mega only) This toggles *no-echo mode*. Normally when you enter a note, it's automatically played so you get some feedback. But in some circumstances this isn't desirable: for example, if you're entering *and* playing notes from the same keyboard, then this default behavior will cause two notes to be played whenever you enter a note. **No Echo** will turn this off.[22]

- **Performance** (Mega only) This brings up the Performance Mode sub-menu. You can select these menu items from any mode (not just Performance Mode), but the impact of your changes will only be felt in Performance Mode:

  - **Keyboard** This menu option lets you specify what happens when you press keys or send other MIDI information during Performance Mode. Your options are:
    - `- - - - -` Played notes are entered into the current track. If the user has just changed the current track, then upon playing the first note the track is first cleared.
    - 0 Play along with the sequencer: MIDI data is routed out the default MIDI Out channel.
    - 1–16 Play along with the sequencer: MIDI data is routed out the channel specified.
    - `TRAN` When you press a key, this transposes sequenced notes (on those tracks which have been set to be transposable).
  - **Repeat Sequence** This menu option specifies how many iterations the sequence should play before it stops. The default is "Forever", but you also have 1, 2, 3, 4, 5, 6, 8, 9, 12, 16, 18, 24, 32, 64, or 128 iterations as options.
  - **Next Sequence** This menu option specifies *which* sequence the sequencer should switch to when it has completed the given sequence. Your options are None, or 0...9.

- **Save** This saves the sequence. You then choose the slot to save in. Slots are displayed by number, plus an `S` (for Save), plus optionally an `R` or `S` (for Recorder or Step Sequencer) if the slot is already used by that application. This letter **blinks** if the file isn't the type of your current application to warn you that you will be overwriting the other application's slot.

- **Options** This brings up the Options menu, which enables you to do...

More Step Sequencer Options Beyond the settings above, the step sequencer responds to a number of options in the Options Menu. See Section 16 for detailed information on each of the options. The Step Sequencer responds to the following options:

Tempo	Note Speed	Swing	Note Length	Transpose (on Mega)	Click
In MIDI	Out MIDI	Control MIDI	MIDI Clock	Volume (on Mega)	

Of particular use are Tempo, Note Speed, Swing, and especially Click.

[22]The Mega also has another closely related feature: if you're in play position mode and you have entered a note slightly before its time, Gizmo won't play it (the first time) when its time comes up, because you just played it yourself. This isn't the case on the Uno: you'll hear a double-played note the first time. There's not enough space on the Uno to add this feature, sorry.

Button Summary The Step Sequencer's buttons can be a bit complex. Here's what they do in each mode. Note that once you're changing the tempo, you can also tap the middle button to adjust the tempo in addition to moving the knob.

Control	*Edit Cursor Mode*	*Play Cursor Mode*	*Performance Mode*
Back Button	Exit	Exit	Leave Performance Mode
Back Button (Long)	Toggle Bypass	Toggle Bypass	Toggle Bypass
Middle Button	Rest	Toggle Mute Track	Toggle Mute Track
Middle Button (Long)	Tie	Clear Track	Toggle Solo
Select Button	Start/Stop	Start/Stop	Start/Stop
Select Button (Long)	Menu	Menu	Menu
Select + Middle Button (Long)	Enter Performance Mode	Enter Performance Mode	Trigger Next Sequence
Left Knob	Change Track	Change Track	Change Track
Right Knob	Change Note Position *or* Enter Play Cursor Mode	Leave Play Cursor Mode	Change Tempo *(Press Back/Select when Done)*
Keystrokes, CC, etc.	Enter at Edit Cursor	Enter at Play Cursor	Enter at Play Cursor *or* Transpose *or* Route Out

Remote Control over CC If you have a remote control surface, you can greatly enhance the Step Sequencer's editing and performance chops by sending CC messages over Gizmo's Control MIDI Channel:

- In any mode you can **jump directly to any track,** clear the current track, and directly edit the length, midi out, velocity, fader, transposability, and pattern of the current track. You can also change a variety of sequencer or note settings and perform edits.
- In Edit Cursor Mode or Play Cursor Mode you can also toggle the mute of any track independently and toggle solo.
- In Performance Mode you can **advance the mute of any track independently** and you can advance the solo. You can also trigger a jump to the next sequence.
- These operations are on top of the CC Control Midi operations (Section 16, see "Control MIDI"), namely start/stop/continue, back button, middle button, select button, bypass, and left/right knob.

CC Number	*CC Value*	*Operation*
64 ... 75	Any	Toggle Mute (or Advance Mute in Performance Mode) for track $n - 63$, that is, track 1...12
76 ... 87	Any	Select Track $n - 75$, that is, track 1...12
88	Any	Toggle Solo
89	Any	Toggle Transposable Track
90	Any	Clear Track
91	Any	Toggle Triggering of Next Sequence
92	Any	Toggle Click Track
93	Any	Mark
94	Any	Copy
95	Any	Splat
116	Any	Move
117	Any	Duplicate
16	0–127	Set Track Note Length (press Select to set, Back to cancel)
17	0–127	Set Track MIDI Out (press Select to set, Back to cancel)
18	0–127	Set Track Velocity (press Select to set, Back to cancel)
19	0–127	Set Track Fader (press Select to set, Back to cancel)
20	0–127	Set Track Pattern (press Select to set, Back to cancel)
21	0–127	Set Gizmo's Tempo (press Select to set, Back to cancel)
22	0–127	Set Gizmo's Transpose (press Select to set, Back to cancel)
23	0–127	Set Gizmo's Volume (press Select to set, Back to cancel)
24	0–127	Set Gizmo's Note Speed (press Select to set, Back to cancel)
25	0–127	Set Gizmo's Play Length (press Select to set, Back to cancel)
26	0–127	Set Gizmo's Swing (press Select to set, Back to cancel)
27	0–127	Set Performance Keyboard Mode (press Select to set, Back to cancel)
28	0–127	Set Performance Repeat (press Select to set, Back to cancel)
29	0–127	Set Performance Next Sequence (press Select to set, Back to cancel)

Note that the Set... CCs transfer to the relevant menu, then act as proxies for the left knob. This means that once you've transferred to a menu, *any* of them can be used to set the menu, along with the left knob CC. This can be confusing.

8 The Note Recorder

The Note Recorder lets you record and play back approximately 64 notes[23] spread over 21 measures.[24] You can save up to two recordings on an Arduino Uno and nine recordings on a Mega.

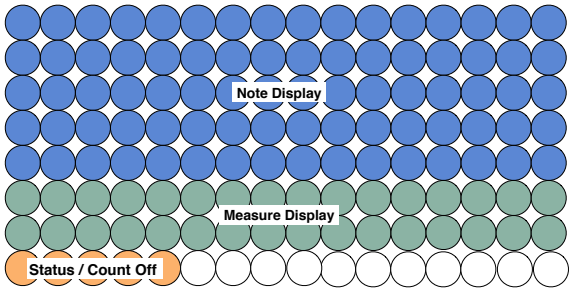


Figure 12: Recorder Display

The Note Recorder screen has two major parts. The *Note Display* section indicates the current note number being played (and hints at the remaining notes available). The *Measure Display* section indicates the current measure being played (and again hints at the remaining measures available). When recording a song, each of these sections will contain a single lit LED indicating the current note or measure position. When playing a song, the Note Display and Measure Display will each also contain an additional lit LED indicating the position of the end of the song.

When you choose the Note Recorder from the root menu, you will be asked to load a recording from a slot. You can do this, or you can create a new empty recording by choosing `- - - -` (meaning *none*). Slots are displayed by number, plus an `L` (for Load) and an optional `R` (for Recorder) or `S` (for Step Sequencer) if the slot already has a file from one of the two applications. This letter **blinks** if the file isn't the type of your current application to warn you that you might eventually overwrite a slot when you save.

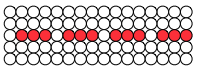


Figure 13: None

After this you will enter the Note Recorder. The Note Recorder is always in one of four states, indicated the patterns at right:

- Stopped
- Playing
- Counting off beats prior to Recording
- Recording

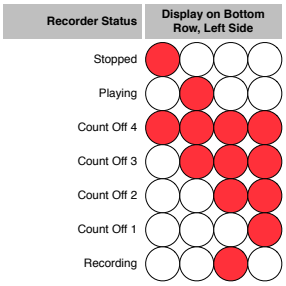


Figure 14: Recorder Status LEDs

When you enter the Note Recorder, the state is initially Stopped. To start playing a song, press the middle button. To stop playing (or stop recording), you also press the middle button. To start recording a song, Long Press the middle button. The recorder will count off four beats for you, then start recording.

You can choose **Repeat Mode** from a menu by Long-Pressing the Select (right) Button. When in repeat mode, the recorder repeats the song it is playing at the measure boundary after it has finished playing the song. Repeat mode also toggles an LED (see Figure 12).

You can save your recorded song by pressing the Select Button. Gizmo will ask you to select a slot to save in, then it will exit the Recorder. Slots are displayed by number, plus an `S` (for Save), plus optionally an `R` or `S` (for Recorder or Step Sequencer) if the slot is already used by that application. This letter **blinks** if the file isn't the type of your current application to warn you that you will be overwriting the other application's slot.

[23] Why *approximately* 64? Because the recorder is recording MIDI NOTE ON and NOTE OFF messages. If you provide 64 pairs of these messages, it'll take up exactly all the available memory. But you could plausibly send a few more NOTE ON than NOTE OFF messages, in which case at the end of the recording all the remaining outstanding notes will be terminated. This would allow you to squeeze in a few (perhaps 16) more notes in rare circumstances. But I'd not rely on it. BTW, if you looked carefully you would have noted that the Note Display in Figure 12 has space for 16 more notes too.

[24] Why 21? Because notes are stored with a timestamp of eleven bits, resulting in $2^{11} = 2048$ timesteps. A single timestep is $1/24$ of a quarter note, and a quarter note is $1/4$ of a measure, so we have $2048/24/4 = 21\frac{1}{3}$ measures. We round that down to 21.

More Recorder Options You can also choose the Options Menu from inside the Note Recorder: it's part of the menu brought up by Long-Pressing the Select button. The Recorder responds to a number of options:

Tempo	Transpose (on Mega)	Volume (on Mega)	Click
In MIDI	Out MIDI	Control MIDI	MIDI Clock

Of particular use to you will be Tempo and Click. Click gives you an audible click track: and the count-in is also clicked for you.

Recording and Quantization The Recorder records nothing but MIDI NOTE ON and NOTE OFF messages. It presently quantizes these messages, rounding them up or down to the nearest MIDI Clock pulse, that is, 1/24 of a quarter note.[25]

[25]This is pretty low-resolution, but there you have it. Space is tight.

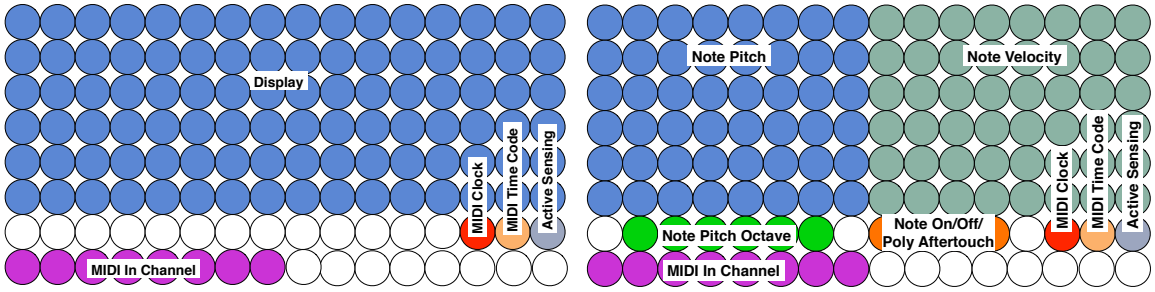


Figure 15: (Left) General Gauge Display, and (Right) Note Gauge Display

9 The MIDI Gauge

The MIDI gauge listens to incoming MIDI messages and displays them on-screen. This only occurs if the MIDI IN channel is something other than NO CHANNEL (See **Options→ In MIDI**, in Section 16).

There are three kinds of MIDI messages for purposes of display. *General* messages take up the whole screen with a single message. *Note* messages split the screen between pitch and velocity of the given note (on, off, polyphonic aftertouch). Finally *Frequent* messages are simply displayed by lighting a specific LED to indicate their presence.

Messages that are associated with a MIDI channel will have their channel number displayed at bottom left (see Figure 15 [Left]). Note that if the MIDI IN channel is something other than OMNI (ALL Channels), then only messages associated with the MIDI IN channel will be displayed.

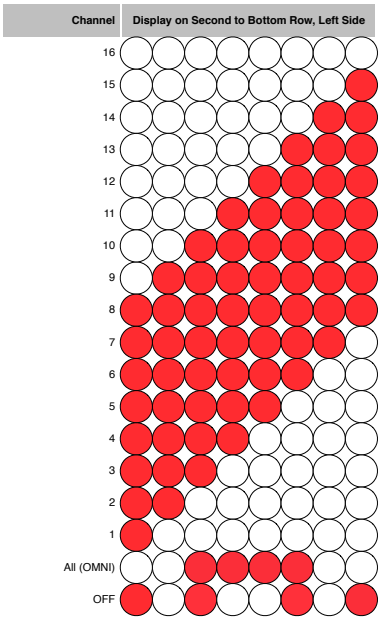


Figure 16: MIDI Channel Values

General Gauge Display These are simple displays which take up the entire screen.

- **Pitch Bend** The 14-bit value is displayed
- **Channel Aftertouch**. `AT` is displayed, followed by the value.
- **Program Change** `PC` is displayed, followed by the value.
- **Control Change (CC)** If the CC parameter is for 7-bit values, then the value is shown first, then text is scrolled with the value, then `CC`, then the parameter. If the CC parameter is for 14-bit values, the MSB of the value is shown first, then text is scrolled with the value, then `CC`, then the parameter, then in parentheses the 14-bit (MSB+LSB) value.
- **Registered Parameter Number (RPN)** The value (7-bit MSB only) is shown first, then text is scrolled with the value (MSB), then `RPN`, then the parameter (14-bit), then in parentheses the 14-bit (MSB+LSB) value.
- **Non-Registered Parameter Number (NRPN)** The value (7-bit MSB only) is shown first, then text is scrolled with the value (MSB), then `NRPN`, then the parameter (14-bit), then in parentheses the 14-bit (MSB+LSB) value.

- **Channel Mode** On the Uno, the channel mode value is shown first, then the text `CHANNEL MODE`, then the channel mode parameter (120–127). The Mega is more specific: the meaning channel mode parameter and value are instead shown as a text message (such as `RESET ALL CONTROLLERS`). In the case of `MONO ON` the channel count is shown first. The Channel Mode parameters are:

Parameter	*Value*	*Meaning (Displayed On Mega)*
120	Normally 0	All Sound Off
121	Normally 0	Reset All Controllers
122	0=Off, 1=On	Local Control Off (*or* On)
123	Normally 0	All Notes Off
124	Normally 0	Omni Off
125	Normally 0	Omni On
126	*C*	Mono On (*with C channels, shown first*)
127	Normally 0	Poly On

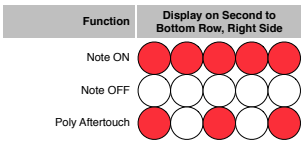


Figure 17: Note Function

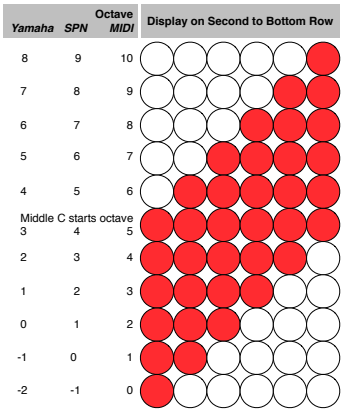


Figure 18: Octave Values. Three common octave numbering schemes shown: Yamaha, SPN (so-called *Scientific Pitch Notation*), and MIDI. SPN is the most common on keyboards.

- **System Exclusive** is *not* indicated: Gizmo does not detect it.
- **Song Position** `SPOS` is displayed.
- **Song Select** `SSEL` is displayed.
- **Tune Request** `TREQ` is displayed.
- **Start** `STRT` is displayed.
- **Continue** `CONT` is displayed.
- **Stop** `STOP` is displayed.
- **System Reset** `RSET` is displayed.

Note messages are displayed with both the **pitch** (at left) and the **velocity** (at right) of the note value, plus the particular kind of message in question: **Note On, Note Off,** and **Polyphonic Aftertouch**.

Note Display How exactly do you distinguish between a Note On, Note Off, or Polyphonic Aftertouch message then? Figure 15 [Right] shows a four-LED region, in orange, where a certain pattern is displayed to indicate this. The patterns are shown in Figure 17.

Note pitches are displayed by showing the note (such as B♭), with the octave value directly below it. MIDI has eleven different octaves (0–10): Middle C is the bottom note of octave 5. The octave number is displayed as shown in Figure 18. As usual, the MIDI In Channel is shown below that.

Frequent MIDI Display These are messages which are too fast to display usefully in most cases. In each case, a specific LED will be toggled (see Figure 15). The messages are:

- **Active Sensing**
- **MIDI Time Code**
- **MIDI Clock**

10 The Controller

The Controller allows you to assign to each of two buttons and four potentiometers any NRPN, RPN, Control Change (CC), or Program Change (PC) value; or (on the Mega) Pitch Bend or Channel Aftertouch.These assignments are stored in Flash memory and survive reboots and power cycling. You can also (on the Mega) set up an eight-stage loopable envelope to do complex control changes, such as emulating the Waldorf Microwave XT's "wave envelope", or doing unusual LFOs. Finally, the Mega also sports a sophisticated Random LFO to do control changes.

The Controller interacts with Bypass mode differently than other applications. In bypass mode, the Controller passes all the incoming MIDI data to MIDI out as usual. But it will still emit values when you press the buttons or more the knobs. This is because it's typical that you'd want to use Controller to (for example) manipulate the VCF on an analog synthesizer while playing it via a remote keyboard. To do this, you'd plug the keyboard into Gizmo, plug Gizmo into the synthesizer, and turn on bypass mode.

Go

This enters the controller proper. Turning the left or right knobs, or pressing the Select or Middle buttons, will cause them to issue MIDI messages as assigned (below). If you press the Back button, you can exit the controller.

When you turn a knob or press a button, and the knob/button has had its control type set to something other than OFF, then the appropriate MIDI message is sent, and the value of the message is displayed as a number on-screen. Messages are only sent if the MIDI OUT channel is something other than NO CHANNEL (See **Options→ Out MIDI**, in Section 16).

Several protocols (NRPN, RPN, some CC) can accept 14-bit numbers, that is, values in the range 0...16383. However Gizmo's potentiometers have a maximum resolution of 10 bits, that is, 0...1023. The potentiometers in fact send 10 bits of data: their top 7 bits are the MSB, and their bottom 3 bits become the upper 3 bits of the LSB (the bottom 4 bits are padded with zeros). While only the MSB is displayed on-screen, this partial LSB is being sent as well to give you a bit more control.

Left Knob and **Right Knob** (`L KNOB` and `R KNOB`)

These submenus let you select your control type and parameter number for the left and right potentiometers.

In each, you are first given the option of what kind of **control type** (OFF, CC, NRPN, RPN, PC, or (on the Mega) Pitch Bend or Aftertouch)you would like to manipulate with the left knob. If OFF, then turning the knob will not do anything.

After you have selected a control type, if you have selected CC, NRPN, or RPN, you will be then asked to select a **controller parameter number**. (Otherwise ,for PC and OFF, you will go back to the top-level Controller Menu). CC parameter numbers may range 0 ... 119. NRPN and RPN parameter numbers may range 0 ... 16383. When you have completed this, you will be sent back to the top-level Controller menu.

Middle Button and **Select Button** (`M BUTTON` and `R BUTTON`)

Thees submenus let you select your control type, parameter number, and on/off values for the middle and select buttons.

In each, you are first given the option of what kind of **control type** (OFF, CC, NRPN, RPN, PC, or (on the Mega) Pitch Bend or Aftertouch)you would like to manipulate with the middle button. If OFF, then pressing the button will not do anything.

After you have selected a control type, if you have selected CC, NRPN, or RPN, you will be then asked to select a **controller parameter number**. (Otherwise ,for PC and OFF, you will go back to the top-level Controller Menu). CC parameter numbers may range 0 ... 119. NRPN and RPN parameter numbers may range 0 ... 16383.

When you have completed this, you will be then asked to enter the value sent when the button is **pressed**. This value must be 0...127 (the Controller does not send 14-bit values). Afterwards, you will be similarly asked to enter the value sent when the button is **pressed again** (the buttons act as toggles). When you have completed this, you will be sent back to the top-level Controller menu.

Wave

This submenu controls the 8-stage Wave Envelope (see Section 10.1 next).

Random

This submenu controls the Random LFO (see Section 10.2).

[Mega Only] **Analog Port 2** and **Analog Port 3** (`A2` and `A3`)

These submenus work exactly like the **Left Knob** and **Right Knob** menus, but they are for additional potentiometers or other devices which you can attach to ports A3 and A3, respectively, on your Arduino. For example, I use them to get information from a joystick.[26]

Warning Gizmo could *theoretically* react to knob changes every 4 MIDI bytes read at full MIDI speed. However an RPN/NRPN command will take up to 12 bytes (or 18 if `INCLUDE_SEND_NULL_RPN` is turned on in the code), and a CC command could take up to 6 bytes. This means that turning a knob may cause Gizmo to generate RPN/RPN/CC messages faster than it can send them. Eventually the Arduino's write-buffer will fill with unwritten messages and the Arduino will pause until it's sent stuff out. This could cause Gizmo to miss an incoming MIDI byte. So if you're sending control commands while (say) at the same time passing data through, this may be an issue: though I have never had it happen to me.

10.1 The 8-Stage Wave Envelope

[Mega Only] The Controller also sports an **8-stage envelope** to control any single control type (CC, NRPN, RPN, PC, Pitch Bend, Channel Aftertouch). This envelope is modeled after the 8-stage "Wave Envelope" found on Waldorf Microwave synthesizers, designed to make cool modulations through a wavetable. For some reason the Waldorf Blofeld lacks this envelope, and I wrote this for Gizmo to give the Blofeld a rough approximation. But you may find it useful in other contexts as well! You can also use this envelope to easily do an LFO with a triangle wave, a sawtooth wave, or a pulse wave, among others.

This is not an ADSR envelope: there is no sustain and no release. Rather, the envelope has eight *stages*, each with a *control value* and a *time interval*. As a key is held down, Gizmo iterates through all of the stages one by one. Each stage starts at the stage's value and lasts for its time interval, gradually interpolating until it reaches the value of the next stage. Your envelope doesn't have to have all eight stages: you can cut it short by setting a stage time interval to **off** (`- - - -`). This has different effects depending on whether you're looping or doing a one-shot envelope (see below).

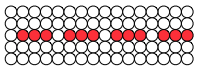
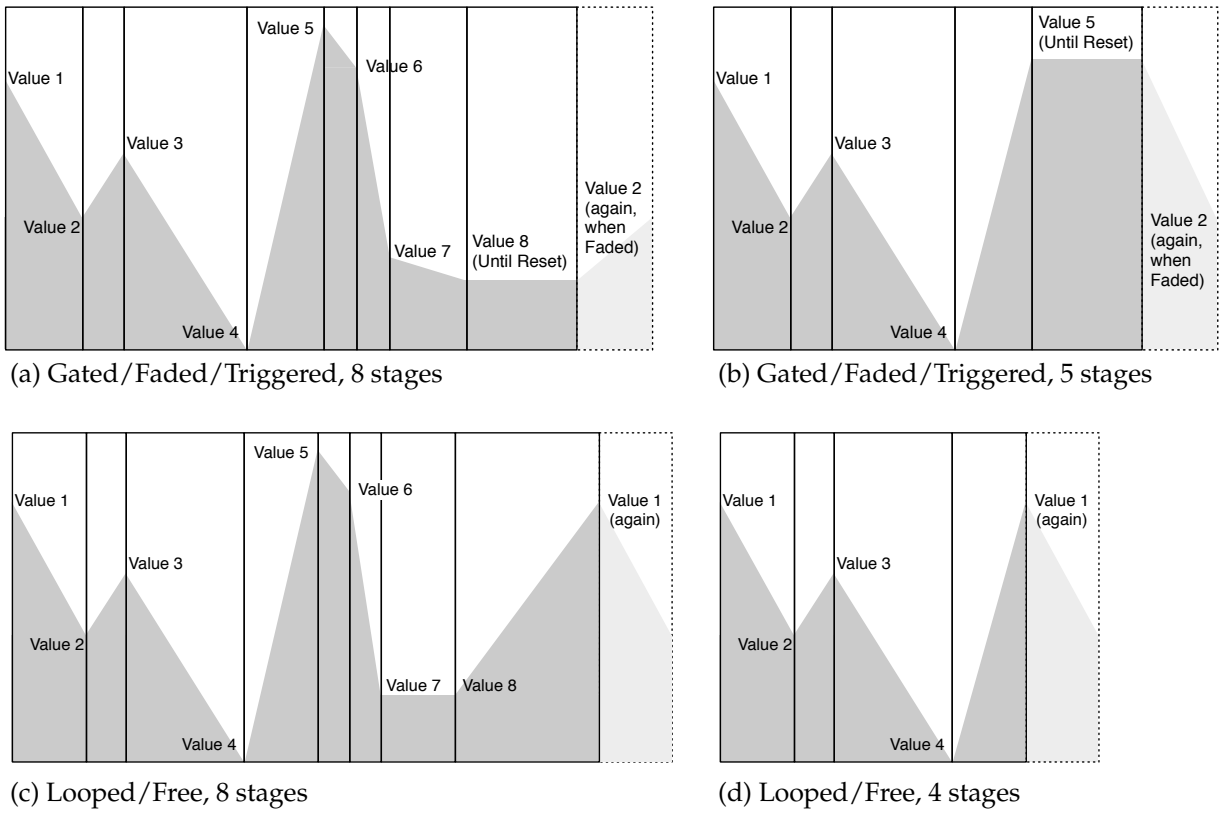


Figure 19: Off

Modes There are five envelope modes:

- **Gated** This is the default. The envelope will begin at the the stage 1 value when a key is pressed, and then start sweeping. An envelope has expired when it reaches (not completes) stage 8, or when it reaches a stage whose time interval is "off". When you release all the keys, or when the envelope has expired, then the control value will stay fixed at its current value until restarted. You restart the envelope by releasing all the keys and then pressing a new one.
- **Faded** This just like Gated, except that after an envelope is restarted, instead of resetting to Value 1 and moving towards Value 2 again, it will start at its previous final value and then move towards Value 2. This prevents the sudden jump to Value 1. This might be useful for multiple long legato pads.

[26] *Why not A4 and A5 too?* Because A4 and A5 are used for I2C, which the screen uses. You could of course add in A6...A12 too.



(a) Gated/Faded/Triggered, 8 stages

(b) Gated/Faded/Triggered, 5 stages

(c) Looped/Free, 8 stages

(d) Looped/Free, 4 stages

Figure 20: Examples of the 8-Stage Envelope in Action.

- **Triggered** This is like Gated, except that the envelope continues even if you have released all the keys. You restart the envelope by releasing all the keys and then pressing a new key. This might be useful for sounds with long releases.
- **Looped** The envelope will begin at the the stage 1 value when a key is pressed, and then start sweeping. An envelope has expired when it *completes* stage 8, or when it *reaches* (not completes) a stage whose time interval is "off". When an envelope has expired, Gizmo loops to stage 1 again. If you release all the keys, then the control value will stay fixed at its current value until restarted. You restart the envelope by releasing all the keys and then pressing a new key.
- **Free** This is like Looped, except that envelope is constantly running and looping at all times. The envelope cannot be stopped or restarted.

Please note that if you're using Gated, Faded, or Triggered, then the time interval for stage 8 serves no function. In Looped or Free, it indicates the length of stage 8 before returning to stage 1.

Values All stage values are 0...127. However the interpolation is 14-bit if you're doing NRPN, RPN, or Pitch Bend. Otherwise the interpolation is 7-bit.

Intervals A stage time interval can any n from 0...254 (or **off** as discussed earlier). The envelope can be **synced** to the MIDI clock (or **unsynced**). When synced, a value of n is a 32nd note, that is, $n/8$ of a beat. So if you want your interval to last a 4-beat measure, set $n = 32$.

When unsynced (the default), n represents approximately $n/16$ steps per second, which is roughly the same as what n is in synced mode when the tempo is 120 BPM. At $n = 254$ (the maximum), the interval length is 15.875 seconds.

Display When Running If the envelope is expired or hasn't started yet, - - - - is displayed. Otherwise it displays two numbers. The left number is the MSB of the current control value. The right number is the stage (1...8).

Menu Options The menu options in the **Wave** submenu are:

- **Go** Start the envelope.
- **Control** Specify the envelope control type. If CC, NRPN, or RPN, additionally specify the parameter number.
- **Envelope** Set the 8 envelope values and time intervals. Remember that an interval can also be - - - -, which indicates that that is the final stage. The final entry in this submenu is **Clear**, which resets all the values to 0 all the intervals to - - - -.
- **Mode** Specify the mode (Gated, Faded, Triggered, Looped, or Free).
- **Sync** *or* **Unsync** Sync the envelope to MIDI clock or unsync it.

Fun with LFOs You can use the Wave Envelope to make a variety of LFOs. For example, you can make Sawtooth, Triangle, and various Square wave LFOs as follows. First set the Mode to Looped or Free. Then set the Envelope values as shown in the table below:

- x is the minimum value of your wave (0...127).
- y is the maximum value of your wave (0...127, $x \leq y$).
- n is the wavelength (a stage time interval), (0...254). So for example if you want a synced LFO to repeat every measure, you could set $n = 32$.

	Sawtooth	*Triangle*	*Square*	*Square, p% Pulse Width*	*Sawtooth → Triangle → Square!*
Value 1	x	x	x	x	x
Length 1	n	$n/2$	$n/2$	$n \times p/100$	n
Value 2	y	y	x	x	y
Length 2	0	$n/2$	0	0	0
Value 3	anything	anything	y	y	x
Length 3	- - - -	- - - -	$n/2$	$n \times (100 - p/100)$	$n/2$
Value 4			y	y	y
Length 4			0	0	$n/2$
Value 5			anything	anything	x
Length 5			- - - -	- - - -	0
Value 6					y
Length 6					$n/2$
Value 7					y
Length 7					0
Value 8					x
Length 8					$n/2$

Hints This envelope isn't like the envelopes in your synthesizer: those are per-voice. This envelope controls *all* the voices playing at the same time — after all, it's just manipulating a single control (perhaps a CC parameter). So if you play a key, then release, and your sound has a long release, then when you play a new key, it'll suddenly change the control value for the old note along with the new one. Thus this might work best with sounds with fairly short releases.

The envelope updates its interpolated values at no more than 100 times a second, so as not to overwhelm the MIDI stream. It's possible that your device can't handle changes that fast. You can change things with the `WAVE_COUNTDOWN` constant found in the file `Control.h`. It's by default set to 32. Increasing it will slow things down.

If the envelope feels stepped or discretized, changing `WAVE_COUNTDOWN` won't help things. Rather, it's likely the fact that you're doing low-resolution CC (7-bit). There's little you can do about that, it's the nature of MIDI.

Last: if you're wondering why you can't select 0 as a length, this is because you're only turning the left knob. Turn the right knob to get finer-grained choices.

10.2 The Random LFO

Gizmo's Random LFO modifies a control value over time. The Random LFO permits both random walks and sample-and-hold style random values of different kinds. In general, it operates by repeatedly choosing a new (random) target value, then over a specified length of time, either (random walk) gradually moving that value towards the target, or (sample and hold) going to the new value immediately. Specifically the LFO has, count 'em, *eight modes*:

- **Gated** This is the default. When a key is pressed and no other keys are down, the LFO chooses a new random start position and new random end position, and sets the control value to the start position. Then over a specified period of time it gradually shifts the control value to the end position. When the time is over, it picks a new random end position, then gradually shifts the control value to that one, and so on. When the last key is released, the LFO stops, and the current control value is maintained.
- **Triggered** This is like Gated, except that the LFO does not stop when the last key is released.
- **Not Reset** [27] This is like Gated, except that when a new key is pressed after all keys are released, the LFO doesn't choose a new random start position; it uses the existing control position. This would be the obvious choice for situations such as the LFO controlling pitch.
- **Free** This LFO is free-running and not affected by keystrokes. The LFO immediately chooses a new random start position and new random end position, and sets the control value to the start position. Then over a specified period of time it gradually shifts the control value to the end position. When the time is over, it picks a new random end position, then gradually shifts the control value to that one, and so on.
- **Sample and Hold (SH) Gated** This is like Gated, except that the control value only changes when we have selected a new random end position.
- **Sample and Hold (SH) Triggered** This is like Triggered, except that the control value only changes when we have selected a new random end position.
- **Sample and Hold (SH) Not Reset** This is like Not Reset, except that the control value only changes when we have selected a new random end position.
- **Sample and Hold (SH) Free** This is like Free, except that the control value only changes when we have selected a new random end position.

There are several parameters which control the randomness of the LFO:

[27] Yeah, it's not a good name.

Length The length of time before selecting a new random value is handled similarly to the stage-interval time in the 8-stage wave envelope. Specifically it can be any value n from 0...254. However, these values are three times faster than in the 8-stage wave envelope (because fast LFOs are more useful, and long time lengths for an LFO are not).

The envelope can be **synced** to the MIDI clock (or **unsynced**). When synced, a value of n is $n/24$ of a beat. So if you want your interval to last a 4-beat measure, set $n = 96$. When unsynced (the default), n represents approximately $n/48$ steps per second, which is roughly the same as what n is in synced mode when the tempo is 120 BPM. At $n = 254$ (the maximum), the interval length is 5.2917 seconds.

Just beyond 254, you can also set n to `HIGH`. This makes the length infinity: until it is reset, the LFO will not change its current value.

Range The degree of randomness in selecting new random values. This is a value n from 1...127. When a new random end position must be chosen, we take the old end position (which will become the new start position) and add a random amount between $-n...n$. This attempt is repeated until we find a valid random end position value. This this ranges from $n = 1$ very gradual random walks to $n = 127$ complete random selection.

Initial Value This is the starting point of the LFO, and is any value 0...127. When the LFO is reset, we choose a new starting point by taking this value and generate a random number within the *range* of this value.

Display and Control When Running If the envelope is expired or hasn't started yet, `- - - -` is displayed. Otherwise it displays two numbers. The left number is the MSB of the current control value. The right number the MSB of the target end position.

You can also dynamically change the current LFO **Length** with the left pot, and the LFO **Range** with the right pot. Additionally, directly beneath each displayed number is a range showing (at a resolution of 16) the current Length (left) and Range (right). When you exit the LFO, the length and range are restored to the values to which they had originally been set (by you).

These two ranges are shown because you can control

Menu Options The menu options in the **Random** submenu are:

- **Go** Enter the LFO.
- **Control** Specify the envelope control type. If CC, NRPN, or RPN, additionally specify the parameter number.
- **Mode** Specify the LFO mode.
- **Range** Specify the LFO range.
- **Initial Value** Specify the LFO initial value.
- **Length** Specify the LFO length.
- **Sync** *or* **Unsync** Sync the LFO to MIDI clock or unsync it.

Hints The longer the length, the less random the LFO will sound. So if you want a blast of random noise, set the length to 0.

Just like the wave envelope, the LFO updates its interpolated values at no more than 100 times a second, so as not to overwhelm the MIDI stream. It's possible that your device can't handle changes that fast. You can change things with the `WAVE_COUNTDOWN` constant found in the file `Control.h`. It's by default set to 32.

Increasing it will slow things down. If the envelope feels stepped or discretized, changing `WAVE_COUNTDOWN` won't help things. Rather, it's likely the fact that you're doing low-resolution CC (7-bit). There's little you can do about that, it's the nature of MIDI. Try increasing your range.

Last: if you're wondering why you can't select 0 as a length, this is because you're only turning the left knob. Turn the right knob to get finer-grained choices.

11 The Keyboard Splitter

[Mega Only] The Splitter allows lets you (1) split your keyboard into two zones, each controlling a different MIDI channel; or (2) split your keyboard into *three* zones, two controlling different MIDI channels and the third (middle) zone playing both MIDI channels simultaneously[28]; or (3) fade or balance your entire keyboard such that if you play a note quietly, one channel will predominate, but if you play loudly, the other channel will predominate.

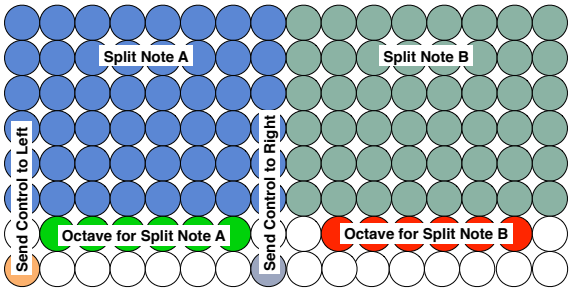


Figure 21: Splitter Layout

The two MIDI channels in question are the Default MIDI Out Channel (defined in the Options, see Section 16), and an Alternative MIDI Out Channel which you specify in the Splitter. If you're splitting your keyboard, you'll also specify up to two notes, **Split Note A** and **Split Note B** (Split Note B is optional), and whether Control Change, Aftertouch, and the like go to the left split region or the right split region.

- If you want to just split your keyboard, specify only Split Note A at the split point. Notes in the left (lower) region will go our the Alternative MIDI Channel, and notes in the right (upper) region will go out the Default MIDI Channel.
- If you want your keyboard to play **both** channels, make Split Note A be the lowest note on your keyboard, and Split Note B be the highest note on your keyboard. Notes played on the keyboard will be sent out both the Alternative MIDI and Default MIDI Channels.
- If you want your keyboard to have **three regions**, a left-only region, a right-only region and a middle region where both are played, make Split Note A be the lower note in the middle region and Split Note B be the higher note. Notes in the left (lower) region will go our the Alternative MIDI Channel, notes in the right (upper) region will go out the Default MIDI Channel, and notes in the middle region will go out both channels.
- Note that it is possible to also give you keyboard three regions where in the middle region **neither** channel is played. This is probably not very useful. But it happens when you define Split Note A to be *above* (to the right of) Split Note B.
- Finally, if you want to fade or balance your keyboard so that notes on the two channels are played with opposite velocities, you simply choose this option (see below). You don't need to specify either Split Note A or B, but you will need to define the alternative MIDI Channel.

Fade Behavior The current fade behavior function is:

$$\begin{aligned} \textit{Default MIDI Velocity} &= \textit{Velocity} - \textit{Velocity} \times (\textit{Velocity} + 1)\,/128 \\ \textit{Alternative MIDI Velocity} &= \textit{Velocity} \times (\textit{Velocity} + 1)\,/128 \end{aligned}$$

This is a fancy way of saying that the total velocity of both channels is the same as the input velocity, and that the ratio of the Alternative velocity versus the Default velocity increases linearly as you get closer to 127. I don't know if this is the right approach, and certainly different keyboards synthesizers have different and often nonlinear velocity mappings. But there you have it. If you'd like to play with it, it's in the function `handleNoteOn(...)` in `TopLevel.cpp`.

As you play notes in the fader, their velocity will appear on-screen.

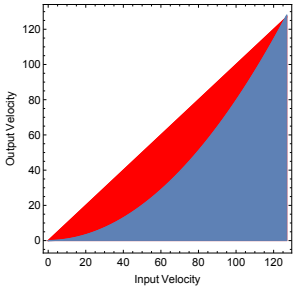


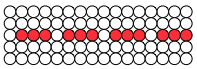
Figure 22: Cumulative Velocity the MIDI Out Channels when fading. Default is red, and Alternative is blue.

[28] This is commonly known as *layering*.

Defining Notes and Channels If you press the Select Button, you'll be asked to enter a note. This note will be Split Note A. By default this note is Middle C. You can back out if you don't want to enter the note.

If you press the Middle Button you'll be asked to enter a note. This note will be Split Note B. By default this note is off. You can back out if you don't want to enter the note. You can also turn Split Note B off again by once again pressing the Middle Button, at which point you'll be presented with `- - - -`. You can also back out if you don't want remove Split Note B.

You define the Alternative MIDI Out channel by Long-Pressing the Select Button. The Alternative MIDI Channel must be a value 1–16 (you can't choose Off).



The Splitter will display your two Split Notes, as shown in Figure 22.

Routing and Fading Long-pressing the Middle Button cycles through three options:

- Split/Layer the Keyboard, and route all CC, PC, Channel Aftertouch, and NRPN/RPN to the *right* (upper) split channel, that is, to the Default MIDI Out channel. Pitch Bend is also routed to the right channel if you're doing a simple split, but if you're layering, it's routed to both channels. Polyphonic Aftertouch is always routed to the channel handling the note in question, as are all Note On and Note Off messages. The screen will show the Split A note, and will light the "left" LED (see Figure 22)..
- Split/Layer the Keyboard, and route all CC, PC, Channel Aftertouch, and NRPN/RPN to the *left* (upper) split channel, that is, to the alternative MIDI Out Channel. Pitch Bend is also routed to the left channel if you're doing a simple split, but if you're layering, it's routed to both channels. Polyphonic Aftertouch is always routed to the channel handling the note in question, as are all Note On and Note Off messages. The screen will show the Split A note at left and the Split B note at right, and will light the "left" LED (see Figure 22).
- Fade/Balance the Keyboard. CC, PC, Channel Aftertouch, and NRPN/RPN is routed to the alternative MIDI Out Channel. Pitch Bend, Polyphonic Aftertouch, all Note On and Note Off messages (though with inverted velocities as discussed earlier) are routed to both channels. The screen will display `FADE`.

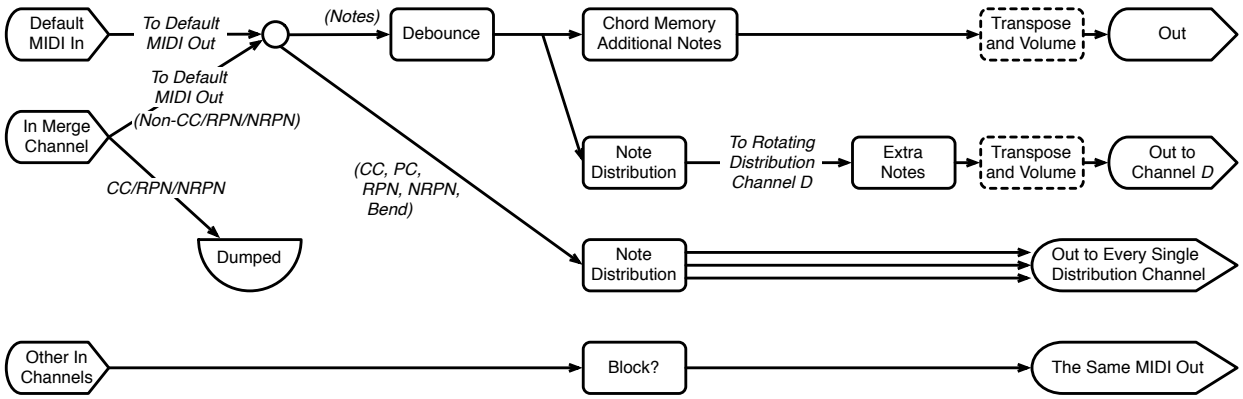


Figure 23: Flow graph of the Thru Facility.

12 The Thru Facility

[Mega Only] The **Thru Facility** passes MIDI through the box, but if MIDI data comes in via certain channels, it applies various optional transformations to it before it sends it out:

- First, you can *merge* two incoming channels: the Default MIDI In channel and a **Merge Channel**. Additionally you can also merge *all* channels by simply setting the Default MIDI In channel to OMNI. Additionally, channels other than Default MIDI and Merge are passed through, unless you *block* them.
- Note data and Control data (CC/PC/RPN/NRPN/Pitch Bend) are then handled differently. We'll start with Note data. Note data is passed through an option that allows you to **debounce notes**. The objective here is to counteract the situation where you hit a drum pad, and instead of staying down, it quickly bounces up, then down again, resulting in a buh-dum sound..[29] It works like this. When you first press a note, Gizmo will require that this note be no less than *N* milliseconds long. Furthermore, if the same note is pressed within these *N* milliseconds, Gizmo will ignore it. Large values of *N* catch more bounces but prevent you from playing the pad rapidly, and will also prevent you from making short stabs. Thus this would be better for one-shot sounds (drums, etc.) which play in their entirety regardless of when the note-off occurs.
- You can then **distribute** incoming notes, one by one, to separate MIDI channels, rotating the distribution round-robin. Thus if you have multiple copies of the same monosynth, you could distribute notes to the various synths and basically treat them as if they were a polysynth.[30]
- Further, you can have Gizmo add **extra notes**, that is, pile up notes you're playing multiple times extremely rapidly. For example, if you play the note G, Gizmo might send four G notes out essentially at the same time. This is useful for fattening up a synth but not making it fully unison (mono). For example, I have an Oberheim Matrix 1000 with six voices. If I instruct Gizmo to provide two extra notes, then every time I press a note, Gizmo emits *three* copies of that note to the Oberheim, causing three voices to play together. This basically turns the Matrix into a fatter two-voice polysynth. **Important Note**: Polyphonic Aftertouch is not repeated.[31]
- You can also perform **chord memory**. Here you first specify a chord, and then afterwards, every time you play a note, Gizmo will instead emit that chord, transposed so that its lowest note corresponds

[29] For example, I have debouncing issues with the Arturia Beatstep.
[30] This is an idea directly stolen from the MIDIPal.
[31] Because it overwhelms the Arduino's serial output, which blocks, causing Gizmo to miss messages, resulting in stuck notes.

with the note you played. This is a classic facility available on a number of synthesizers, including early ones such as the Oberheim OB-Xa, Sequential Circuits Prophet 600, Korg PolySix, and so on.[32] **Important Note**: Polyphonic Aftertouch is not chorded.[33]

- Control Data (CC/PC/RPN/NRPN/Pitch Bend) follows a different path: a copy is sent to every channel that Gizmo is presently distributing to.
- CC/RPN/NRPN from the Merge Channel is neither merged nor passed through, but is simply ignored. This is because it is nontrivial to merge it.
- Data from channels other than Default MIDI In and Merge is simply passed through. **Warning.** If your Default MIDI Out is not the same as your Default MIDI In (that is, you're routing from one channel to another)
- Finally, if you are distributing incoming notes, control data will also be **distributed**, but instead of sending it round-robin to each of the relevant channels, control data is sent to **all of them in parallel**.

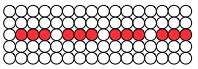
I might add more unusual transformers later. Anyway, here are your options:

Go

Once you have set up your Thru options (see below), if you choose **Go**, the Thru facility starts and you can start playing. You'll see the word `PLAY` on the LED.

Extra Notes

This lets you select how many *additional* simultaneous note-on (and note-off) messages Gizmo should send down a given channel in response to you playing a note.

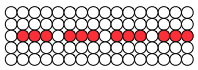


Choose: No Extra Notes, or 1...31

Distribute Notes

This lets you select the number of MIDI channels, beyond the Default MIDI Out channel, over which to distribute notes. Let's say you selected 2, and the Default MIDI Out channel was M. This means that three MIDI channels will be used: M, $M+1$, and $M+2$. This wraps around: if, say, $M = 15$, then it wraps around: the channels used would be 15, 16, and 1).

If you played the five notes A, B, C, D, E, Gizmo would send A down channel M, then send B down channel $M+1$, then send C down channel $M+2$, then send D down channel M again (wrapping around), then E down channel $M+1$ again, and so on. This would let you essentially use Mono devices listening in on the three MIDI Channels as a 3-voice polyphonic device.



Choose: No Extra Channels, or 1...15

Chord Memory or **No Chord Memory**

This will show the text `CHRD`, and you can then specify a chord (of up to eight notes). Thereafter if you play a single note, the chord will be played instead, transposed so that its root note corresponds with the note you play. (If you play more than one note, more than one chord will be played). If you have already chosen a chord, then selecting this option again will turn off chord memory.

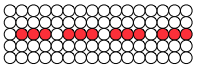
Choose: a chord of up to eight notes.

Debounce

This lets you choose the number of milliseconds between successive identical MIDI notes (the N value discussed earlier). 100 milliseconds seems to work well for me.

[32] This idea was also directly stolen from the MIDIPal.
[33] See Footnote 31.

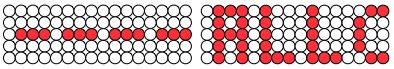
Choose: No Debouncing, or 1...255 (milliseconds)



Merge Channel

Here you select the alternate MIDI-In channel (if any) to merge with the Default MIDI In channel

Choose: No Channel, Channels 1...16, or ALL Channels (OMNI)



Block Others or **Unblock Others**

This lets you choose whether MIDI data on channels other than the Default MIDI In channel and the alternate MIDI-In channel should be blocked entirely or passed through unaltered.

13 Synthesizer MIDI Helpers

[Mega Only] Synthesizers often have weak spots in their MIDI implementations. This application contains various submenus, one per synthesizer, designed to provide an assist for that particular model.[34]

A Note on Performance Often these MIDI helpers are converting CC to NRPN, or converting NRPN to Sysex, and in the process they're emitting something bigger than they're getting (for example, an NRPN message could be between 3 and 5 times larger than a CC message). This is a problem. For example, if it receives CC at a very fast rate, Gizmo won't be able to send out corresponding NRPN fast enough and eventually Gizmo's serial input buffer will overflow. To counter this, Gizmo sends out messages at a fixed rate (typically about 100 per second), and drops some on purpose if they arrive faster than that. Gizmo will always send the most recent packet. This is determined by some constant `SYNTH_NAME_COUNTDOWN` in the appropriate file `synth/SynthName.h` (lower values are faster rates).

13.1 Waldorf Blofeld Module / Keyboard

The Blofeld exposes parameters through CC, not NRPN; this means that not all of its parameters can be accessed (there too many of them). Gizmo maps NRPN messages to corresponding Blofeld Sysex parameter change messages, which allows you to manipulate all of the Blofeld's parameters over NRPN. The NRPN parameter numbers in question are 0...382 (MSB + LSB), corresponding to the Sysex parameters shown in Section 3.1 ("SDATA - Sound Data") of the text file found in `docs/synth/WaldorfBlofeldSysex.txt` in the Gizmo distribution. Values are MSB only. Additionally, changing the value of NRPN parameter number 400 will set the buffer which these parameters effect, namely, 0 (Sound Mode Edit Buffer or Multi Instrument Edit Buffer 0), or 1–15 (Multi Instrument Edit Buffers 1...15). The default is 0.

Display If you are setting the ID, then `ID` is displayed, along with the ID value. Otherwise, just the set value is displayed.

Caveats While you are sending NRPN this way, you can also send some CC messages, but not for CC numbers 6, 32–63, or 98–101: you'll need to send the NRPN equivalents for those. The Blofeld seems to drop some Sysex packets if you send them too fast. Significantly increasing `WALDORF_BLOFELD_COUNTDOWN` helps things but at the cost of lag and discretization. It's probably better to just turn your encoder dial more slowly.

13.2 Kawai K4 / K4r

This code is not yet tested The K4 is only accessible via Sysex. Gizmo maps NRPN messages to corresponding Kawai K4 Sysex parameter change messages, which allows you to tweak all of the K4's (non-Multi) parameters over NRPN. The K4 has 88 parameters, broken into Single (0–69), Drum (70–81), and Effect (82–88) parameters. Single parameters require that you first specify the *source* via NRPN parameter 100: source 1 is value 0, source 2 is value 1, source 3 is value 2, and source 4 is value 3. Drum parameters require that you first specify the *drum key* via NRPN parameter 101: the key values are 0–60, left to right. And Effect parameters require that you first specify the *submix / output channel* via NRPN parameter 101: channel values are 0–7.

For more information, see the PDF file `docs/synth/KawaiK4CorrectedMIDIImplementation.pdf`. In particular, look at to the breakout of parameters as specified in Section 5-6 and the parameter list proper in Section 6 (pay attention to the "Parameter No" column). Note that one parameter (#17) combines four mute values and vibrato shape into a single number. This means you can't change one of them without changing the others. Gizmo handles this by letting you set each of them separately, and it keeps track of what settings you used previously. Set all of them one time each, then as you continue to set them it'll update with the old

[34] Why *these particular models?* Because I own them. If you'd like me to write a helper for another machine here, send me a synth!

values of the others. Mutes 1–4 are set with NRPN parameter 103–106, and Vibrato Shape is set with NRPN parameter 107. See Parameter #17 in Section 6 of the file for appropriate values of these.

Display If you are setting the Vibrato Shape or Mute 1...4, then `VS`, `M1`, `M2`, `M3`, or `M4` are displayed respectively, along with the value. If you are setting the Source, Drum, or Effects Parameter, then `S`, `D`, or `FX` are displayed respectively along with the value. Otherwise, just the value is displayed.

13.3 Oberheim Matrix 1000

This code is not yet tested This machine is dramatically opened up by new firmware (Gligli's v1.16, and Bob Grieb's v1.20), which adds quasi-real-time parameter changes and NRPN support. If you don't have this firmware yet, you should install it.[35] Unfortunately few DAWs support NRPN. So Gizmo lets you map raw CC messages to their NRPN counterparts. The mapping work as follows. The Matrix 1000 has parameters 0–99. To send the Matrix 1000 an NRPN parameter p from 0–53, you send a CC parameter of $p + 10$ (so this ranges 10–63). To send an NRPN parameter p from 54–99, you send a CC parameter of $p + 20$ (so this ranges 74–119). If you send a CC parameter 0–10, 64–73, or 120–127, it is passed through as a regular CC message. To understand the 99 (actually somewhat fewer) parameters, find an online copy of the Matrix 6 (*not* Matrix 1000) manual.[36] Also note that because many Matrix 1000 parameter values are signed (such as -64...63), the NRPN values are implemented unsigned (0...127) in the machine as follows. To send a value v, you should instead send $v + 64$. For example, to send 15, you should actually send 15 + 64 = 79, and to send -3, you would send -3 + 64 = 61. One exception: for parameter number 21 (VCF Initial Frequency), you should just send the number as-is.

Display Just the set value is displayed.

13.4 Korg Microsampler

Anyone who owns one of these machines knows that its MIDI is a disaster. Pitch Bend is steppy, only certain notes can be played, if MIDI START is received the pattern sequencer immediately starts up, and if the device is set to emit MIDI clock (via "auto" or "internal") then MIDI clock pulses are sent but not MIDI START, STOP, or CONTINUE. These last two items can be overcome with judicious use of Gizmo's MIDI clock filter. But on top of it all, the Microsampler's parameters can only be controlled via sysex,[37] and it's entirely undocumented.

Gizmo's helper for the Microsampler converts NRPN to sysex. There are four basic kinds of parameters. First there are *bank parameters*, such as BPM. These are changed straightforwardly with one NRPN command each. Second, there are *pattern parameters* such as Pattern Length. These require a pattern number in addition to the parameter number. To change these, you either first set the pattern number n via NRPN 50 and then set the parameter via its NRPN p, or you can change the parameter directly by changing NRPN number $p + 100 \times n$. Third there are *sample parameters* such as Sample Level. Similar to pattern parameters, these require a sample number in addition to the parameter number, and similarly, you can either first set the sample number n via NRPN 51 and then set the parameter via its NRPN p, or you an change the parameter directly by changing NRPN number $p + 100 \times n$.

Finally, there are *FX parameters*. These require not two but *three* variables: the FX type, the FX Parameter Number, and the parameter value proper. To set these, you first set the FX Type (NRPN 31). This switches the FX Type on the machine. Then you either set the FX parameter number n via NRPN 52 and then set the

[35] I strongly suggest you install Bob Grieb's, it's only $30. http://tauntek.com/Matrix1000Firmware.htm By the way, if you have a Matrix 6 or 6R, check out http://tauntek.com/Matrix6Firmware.htm though it doesn't support NRPN.

[36] You can also see the parameter list and MIDI implementation at http://www.youngmonkey.ca/nose/audio_tech/synth/Oberheim-Matrix1000.html Pay attention to the section "MATRIX-1000 SINGLE PATCH DATA FORMAT", particularly the "Parameter" column.

[37] And not all of them: you can't change the start or end trim positions, for example. Furthermore, no *events* can be controlled by sysex. So there's no sysex message to tell the Microsampler to start recording, or to normalize the sound.

parameter via its NRPN p, or you can change the parameter directly by changing NRPN number $p + 100 \times n$. Note that the number of FX parameters varies depending on the FX type chosen.

The FX type can be set independently, as can the parameters emitted by Control 1 and Control 2 (via NRPN 32 and 33 respectively). Note that the parameters emitted by these are a subset of possible parameters for a given FX type.

The full sysex documentation to the best of my understanding, plus the Gizmo NRPN parameter mapping, can be found in the file `docs/synth/KorgMicrosamplerSysex.txt`.

Display If you are setting the Sample Number, Pattern Number, or Effects Parameter Number, then `S`, `P`, or `FX` are displayed respectively, plus the number. Otherwise only the set value is displayed.

13.5 Yamaha TX81Z

This code is not yet tested, and has known bugs The TX81Z's parameters can only be changed via sysex. Gizmo maps NRPN messages to corresponding TX81Z Sysex parameter change messages, which allows you to tweak all of them over NRPN. One oddity in Yamaha's Sysex is that it has MIDI channels. Gizmo will set this channel to Gizmo's MIDI Out channel.

The TX81Z has a great many parameters, split into nine different categories, each assigned a number c as shown: (0) *VCED*, (1) *ACED*, (2) *PCED*, (3) *Remote Switch*, (4) *Octave Micro Tuning*, (5) *Full Micro Tuning*, (6) *Program Change Table*, (7) *System Data*, and (8) *Effect Data*. In all cases, to set a parameter number p from a given category c, you choose NRPN number (MSB+LSB) $c \times 200 + p$. The parameter numbers for categories VCED, ACED, PCED, and Remote Switch are shown on pages 71–73 of the file `docs/synth/YamahaTX81ZManual.pdf`. System Data and Effect Data are set similarly, but the meaning of the parameters is unknown to me. Values are 7-bit (MSB) only.

For Octave and Full Micro Tuning, the parameter number p is the key number. The value of this number is 14-bit (MSB+LSB), where the MSB is the note to assign the key and the LSB is a fine-tuning of the note. Gizmo deviates from the TX81Z standard here to make programming easier: the fine-tuning ranges from 1...63, where 1 is -31, 32 is 0, and 63 is +31.

Because the TX81Z has more than 128 patches, Program Change Table lets you (weirdly) map PC commands (0–127) to specific patches (0–184). See page 68 of the file for information on how these 185 patches are organized. In NRPN, the parameter number p specifies the program change number, and the value (MSB+LSB) is the patch number.

Display Just the set value is displayed.

Caveats The TX81Z doesn't respond well to fast parameter changes. Significantly increasing the value of `YAMAHA_TX81Z_COUNTDOWN` may help this but at the cost of lag and discretization. It's probably better to just turn your encoder dial more slowly.

14 The Measure Counter

The measure counter[38] is basically a stopwatch which counts elapsed beats, measures (bars), and phrases (up to 127 phrases). Alternatively it can count eighths of a second,[39] seconds, and minutes (up to 127 minutes) via the internal clock or external MIDI clock. If the minutes or phrases exceed 127, then `HI` is displayed instead.

To change what a measure (bar) or phrase means, long-press the Select button to bring up the Menu. Here you can choose the **Beats Per Bar** (a value from 1...16)[40] and the **Bars per Phrase** (1...32), and as usual, call up the **Options Menu** to change the **Tempo**, etc.

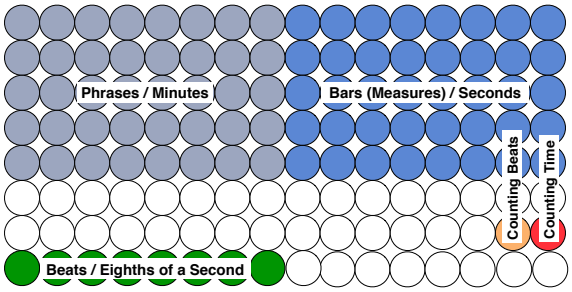


Figure 24: Measure Counter Layout

Pressing the Middle Button starts and and restarts the stopwatch. Long-pressing the Middle Button pauses the stopwatch (pressing the Middle Button afterwards will continue). Additionally, external MIDI Start, Stop, and Continue commands will affect the stopwatch: Stop will pause it, Continue will continue from last pause, and Start will reset the stopwatch and start it fresh.

Normally the measure counter shows beats, measures, and phrases. But if you press the Select button, Measure will instead start displaying elapsed time, so it's basically a stopwatch. More specifically, rather than counting beats, measures (bars), and phrases, Measure will count eighths of a second, seconds, and minutes. The response to buttons is the same, as is the response to MIDI clock directives.

You can also control the Measure Counter via MIDI: if you send a MIDI Play, it will reset the Measure Counter. Note that if you send a MIDI Stop, and Gizmo is listening to external MIDI clock, then the Measure Counter will be halted (and MIDI Continue will continue the Counter). However sending a MIDI Stop will not presently halt the stopwatch.

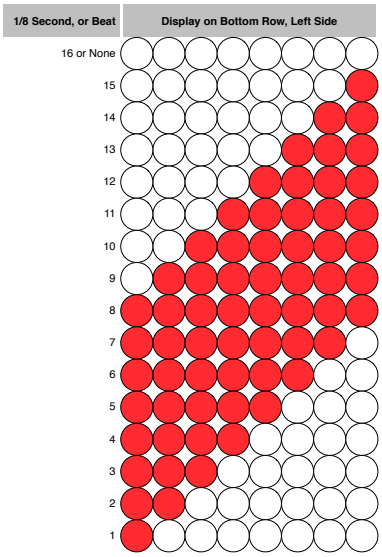


Figure 25: Beats or 1/8 Second Ticks

[38] The Measure Counter was added to Gizmo at the request of `Inkog`, a member of http://gearslutz.com and early adopter of Gizmo.
[39] Why not tenths of a second? Because $10 \times 60 \times 128$ is larger than 2^{16}, so it won't fit in a 16-bit integer.
[40] Why not 32, like Bars Per Phrase uses? Because $32 \times 32 \times 128 > 2^{16}$. So basically for the same reason as Footnote 39.

15 The Sysex Dump Facility

The sysex facility allows Gizmo to load and emit sysex dumps of individual arpeggios or of slots (which can hold either step sequences or note recordings).

The facility is simple. First, you're asked to choose whether you want to upload/download **Slots** or **Arpeggios**. After this, you're asked which slot or arpeggio **number** you want to upload/download. After you have selected one, the sysex facility enters its **transfer mode**.

Initially the transfer mode will display `- - - -` indicating that nothing has happened yet. When you press the **Select Button**, Gizmo will do a sysex dump of your requested slot or arpeggio. You can send this dump as many times as you like. Correspondingly, if you send a slot or arpeggio to Gizmo while in transfer mode, Gizmo will automatically read and save it. (Note that Gizmo will only save an arpeggio if you had selected an arpeggio, and likewise only save a slot if you had selected a slot: otherwise it will display `SYSX` on receiving the wrong type sysex dump).

After either sending or receiving a sysex dump, Gizmo will display an ever-increasing integer. The value of this integer means absolutely nothing: it exists so that when it changes you can tell that something was just successfully received or sent. If Gizmo receives a bad sysex dump, it will display `FAIL`.

Gizmo's sysex dump format is simple. Bytes 0–6 are as follows: `0xF0 0x7D 'G' 'I' 'Z' 'M' 'O'`. Byte 7 is the version number (presently `0x00`). Byte 8 is the sysex type (`0x00` for slots, `0x01` for arpeggios). Next, Gizmo provides a memory dump of the bytes forming a single slot or arp structure (`struct _slot` or `struct _arp` in Gizmo's code). Each byte in this memory dump is nybblized into two bytes in the sysex stream: that is the high 4 bits of the byte are sent first, followed by the low 4 bits. For example, if a memory byte was binary 01101001, then the sysex would contain the two bytes 0000**0110** and 0000**1001** in that order. Next, Gizmo has a checksum, which is simply the low 7 bits of the sum of all the nybbles. Finally comes `0xF7`. This structure is not really meant to be parsed, though a cursory reading of the Gizmo code should make it clear what the memory bytes mean.

Installation Note The Sysex facility is not enabled by default, because in order to use it you also need to make a small modification to the MIDI library. You can't use the Sysex facility on the Uno because there's not enough RAM. See the `All.h` file for installation instructions.

16 Options

Options sets global parameters for the device. These parameters are stored in Flash memory and survive a power cycle. Some options can be also accessed from certain other applications as a convenience.

Tempo

If Gizmo is following its own internal clock rather than relying on an external MIDI clock , this specifies how fast a quarter note is. (See **Options→MIDI Clock**). When not in **Bypass Mode**, the tempo is shown by the pulsing on/off of the **Beat / Bypass Light** (see Figure 2).

Choose: 1 ... 999 Beats Per Minute. Note that this is a large number, and so may require you to choose it with the left potentiometer, then fine-tune it with the right potentiometer.[41]

Alternatively: Tap the Middle Button. The BPM will be set to the rate you tap.

Note At the borders (near 1 or 999) you may find it harder to precisely set the BPM due to anti-potentiometer noise code in Gizmo. Just use the right knob to fine-tune your value.

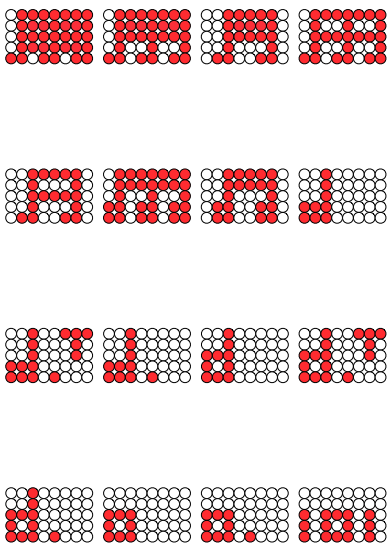
Note Speed

Various applications (arpeggiators, step sequencers) produce notes at a certain rate relative to the tempo. For example, though the tempo may specify that a quarter note is set to 120 Beats Per Minute, the arpeggiator might be generating eighth notes and so is producing notes at twice that speed. You specify the note speed here.

Note speed is shown by pulsing the **Note Pulse Light** (Figure 2).

Choose:

Eighth Triplet (Triplet 64th Note)	1/24	Beat
Quarter Triplet (Triplet 32nd Note)	1/12	Beat
Thirty-Second Note	1/8	Beat
Half Triplet (Triplet 16th Note)	1/6	Beat
Sixteenth Note	1/4	Beat
Triplet	1/3	Beat
Eighth Note	1/2	Beat
Quarter Note	1	Beat (duh)
Quarter Note Tied to Triplet	1 1/3	Beats
Dotted Quarter Note	1 1/2	Beats
Half Note	2	Beats
Half Note Tied to Two Triplets	2 2/3	Beats
Dotted Half Note	3	Beats
Whole Note	4	Beats
Dotted Whole Note	6	Beats
Double Whole Note	8	Beats



Swing

Swing, or **syncopation**, is the degree to which every other note is delayed. 0% means no swing at all. 100% means so much swing that the odd note plays at the same time as the next even note. That's a lot.

Choose: 0% ... 100% (larger percentages are more swing, that is, longer delay every other note)

[41] Gizmo can go lots faster than 999: in theory it could go clear to 31200 or so. But then your synthesizer would explode and we wouldn't want that.

Swing is only applied if the Note Speed is set to thirty-second, sixteenth, eighth, quarter, or half-notes.

Mega Only: **Transpose**

Here you can state that any notes generated by Gizmo be **transposed** up or down by as much as 60 notes. If a note is transposed to the point that it exceeds the MIDI range, it is not played. When appropriate, some applications (such as the MIDI Gauge) ignore this option.

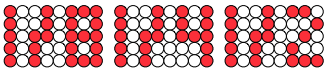
Choose: -60 ... 60

Mega Only: **Volume**

Like Transpose, here you can state that any notes generated by Gizmo have their volume changed by multiplying their MIDI note velocity by some value. If a resulting MIDI note velocity exceeds the maximum (127), it is bounded to 127.

Choose:

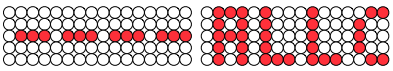
1/8
1/4
1/2
1 (default)
2
4
8



In MIDI [42]

Many applications expect notes and other controls to come in via a specific input MIDI channel. You specify it here.

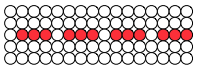
Choose: No Channel, Channels 1...16, or ALL Channels (OMNI)



Out MIDI

Many applications emit notes etc. via a specific output MIDI channel. You specify it here. Other applications can emit notes on several different channels, in which case this value determines the default channel used.

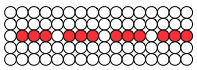
Choose: No Channel, or Channels 1...16



Control MIDI

Gizmo can be controlled via NRPN or (on the Mega) CC messages. You specify the channel here on which it will listen for them.

Choose: No Channel, or Channels 1...16



The default[43] CC and NRPN Messages are (note that CC 14 and 15 are 14-bit: their LSB are sent with 46 and 47):

[42] *Why aren't these called MIDI In and MIDI Out?* Because then they'd be indistinguishable on the menu screen before the text started scrolling.

[43] If you need to change these parameters, see constants (such as `CC_LEFT_POT_PARAMETER`) in the file `MidiInterface.h`. Remember, you'll want the Pot parameters to be 14-bit.

Parameter CC	NRPN	*Min Value*	*Max Value*	*MSB/LSB*	*Description*
14	0	0	1023	MSB + LSB (46)	Turn the Left Potentiometer
15	1	0	1023	MSB + LSB (47)	Turn the Right Potentiometer
102	2	0 (released)	Not 0 (pressed)	MSB	Press/Release the Back Button
103	3	0 (released)	Not 0 (pressed)	MSB	Press/Release the Middle Button
104	4	0 (released)	Not 0 (pressed)	MSB	Press/Release the Select Button
105	5	Any Value	Any Value	MSB	Toggle Bypass
106	6	Any Value	Any Value	MSB	Unlock Potentiometers. When an NRPN message is received, normally the on-board potentiometers are locked so that turning them has no effect. Pressing an on-board button unlocks them: so does sending this NRPN message.
107	7	Any Value	Any Value	MSB	Start Clock. If Gizmo's clock is stopped, resets and starts it. When appropriate, sends a MIDI START message.
108	8	Any Value	Any Value	MSB	Stop Clock. If Gizmo's clock is playing, stops it. When appropriate, sends a MIDI STOP message.
109	9	Any Value	Any Value	MSB	Continue Clock. If Gizmo's clock is stopped, continues it. When appropriate, sends a MIDI CONTINUE message.

These operations are in addition to those discussed in the Step Sequencer (Section 7).

You can also start/stop the clock via by Long-Pressing the Middle button while in the Options menu. And you can start/continue the clock by Long-Pressing the Select button while in the Options menu. See also **Starting, Stopping, and Continuing the Clock** in Section 17.2.

MIDI Clock

Gizmo can respond to a MIDI clock, ignore it, or emit its own MIDI clock. The use of the **Options→Tempo** setting will depend on the setting chosen here.

Choose:

Ignore	Use an internal clock but let any external MIDI clock pass through.
Use	Use an external MIDI clock and also let it pass through.
Consume	Use an external MIDI clock but don't let it pass through.
Generate	Use an internal clock and emit it as a MIDI clock. Don't let any external MIDI clock pass through.
Block	Use an internal clock but don't emit it as a MIDI clock. Don't let any external MIDI clock pass through.

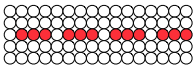
You can also start/stop the clock via by Long-Pressing the Middle button while in the Options menu. And you can start/continue the clock by Long-Pressing the Select button while in the Options menu. See also **Starting, Stopping, and Continuing the Clock** in Section 17.2.

Divide

If your MIDI Clock option is set to **Use**, **Ignore**, or **Generate** (that is, something which causes Gizmo to emit MIDI Clock), then you have the option of slowing down the emitted MIDI clock by *dividing* it by some constant N For example, if your clock is presently running at 120 BPM, and you're doing Generate, and $N = 3$, then Gizmo's emitted MIDI clock will be at 40BPM. Or if you're doing Use and

the incoming MIDI Clock is at 100BPM, and $N = 2$, then Gizmo will adjust this to 50BPM when sent out. Note that Gizmo's internal applications will use the original clock speed. Note that **Bypass Mode** disables all this nonsense.

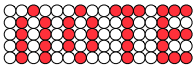
Why is this useful? Here's an example scenario. You have a sequencer on Device A which is playing a song at 120 BPM, and you want to slave another sequencer, on Device B, to it which plays its part at half that speed. You can do this by having Device A send to Gizmo, and Gizmo send to Device B with a divided clock.



Choose: 2 ... 16 (the value for N) when using **Divide**, or None

Click or **No Click**

This toggles the note played by Gizmo to provide a click track for applications such as the Step Sequencer or Recorder. If you select **Click**, you will be asked to play a **note**. The note pitch and velocity form the click. If you select **No Click**, this turns off the click track.



Screen Brightness

This changes the brightness of the LED matrix.

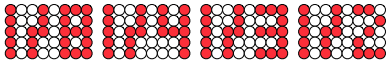
Choose: 1...16 (higher values are brighter)

Mega Only: **Menu Delay**

Gizmo often will display text (such as `SWING`) by filling the screen with the first few letters (in this case, `SWI`), then *delaying* for a certain interval, then scrolling through the whole word horizontally. You can specify how long that delay is.

Choose:

0	Seconds	(scroll immediately)
1/8	Seconds	
1/4	Seconds	
1/3	Seconds	
1/2	Seconds	
1	Second	(default)
2	Seconds	
3	Seconds	
4	Seconds	
8	Seconds	
∞	Seconds	(never scroll)



The About Screen

Presently says: `GIZMO V5 (C) 2017 BY SEAN LUKE`

17 About the Clock

17.1 The Internal Clock

The Arduino has an internal clock with 32 bits, yielding a maximum of 42,949,672,95 ticks (one tick per microsecond) before it rolls back over to 0. This sounds like a lot, but it's not: it's a bit over an hour.[44] This means that once an hour, Gizmo might do something odd, if you're using the internal clock. Most likely it could drop a single pulse (1/24 of a quarter note), so it might be off by that much with respect to other instruments synced to it. You've been warned.

17.2 Starting, Stopping, and Continuing the Clock (on the Mega)

You can start, stop, and continue the clock via NRPN/CC as discussed earlier (see **Control MIDI** in Section 16). But there's another "secret" way to do it on the Mega, using Gizmo's buttons. If you are in the Options menu, long-press the Middle Button: this toggles **starting** or **stopping** the clock. And if you long-press the Select button, you can toggle **continuing** or **stopping** the clock. Remember that you can also get to the Options menu from inside the Arpeggiator, Step Sequencer, or Recorder.

Neither long-pressing the Middle Button, nor using NRPN to start/stop/continue, will do anything if the MIDI Clock is set to **Use, Divide,** or **Consume** (in which case the MIDI Clock is out of your control). See **MIDI Clock** in Section 16.

Note that starting and stopping the clock doesn't mean that an application starts or stops playing. However, the Start message *will* cause applications to reset themselves to the beginning when appropriate. This means that if you want to start/stop/pause/continue the Step Sequencer (for example), you can do it as follows: first stop the clock, then start the sequencer playing. Note that it won't play because there's no clock pulses. But now you can start, stop, and pause/continue the sequencer by controlling the Start/Stop/Continue clock messages directly.

Manually stopping and starting the clock can produce results which look like bugs, but aren't, so you have think about the effects. For example, if you're in **Generate Clock** mode and you've manually stopped the clock, and you go into the Step Sequencer and start playing, nothing will happen. More than once this has confused me enough to look through the code thinking I'd written an error.

[44] 71.5827882666 minutes to be exact.

18 Development Status

		Testing		
Item	*Coded*	*Cursory*	*Significant*	*Known Bugs and Issues*
Arpeggiator	✓	✓	✓	
Step Sequencer	✓	✓	✓	Occasionally the beat doesn't match up with the play cursor; this particularly happens under Swing. I've not been able to track it down.
Recorder	✓	✓	✓	Rarely a note will get stuck after recording: appears that a NOTE OFF message isn't getting matched with the ID of the corresponding NOTE ON message when entered into memory. I also think there may be a bug causing chords near the start of the song to be rushed a bit. Perhaps timing is being delayed for some reason?
Gauge	✓	✓		
Controller	✓	✓		Aftertouch untested. Pitch Bend displays correctly but MIDI output is untested.
NRPN Control	✓	✓	✓	
Transposition / Volume	✓	✓	✓	
Brightness / Delay	✓	✓	✓	
Tempo / Note Speed	✓	✓	✓	
MIDI In/Out Channels	✓	✓		No test of OMNI yet
Bypass	✓	✓	✓	
Display Subsystem	✓	✓	✓	
Timing Subsystem	✓	✓	✓	
Interface Utilities	✓	✓	✓	
Storage Facility	✓	✓	✓	
MIDI Clock Control	✓	✓		Odd things may happen when changing the clock control option while an external clock is active.
Clock Division	✓	✓		
Keyboard Splitting	✓	✓		Works well but the fade option is perhaps a bit unnecessary and weird.
Thru Facility	✓	✓		
Tap Tempo	✓	✓		Seems to work, but might be better to average in tempos. This would require floating point probably, which would be too big for the Uno.
Measure Counter	✓	✓		Seems to work. MIDI clock response has not been tested.

19 Development Hints

So you want to write a Gizmo application? Here's some hints for you.

19.1 Notes

The code is largely written in C, with a few little C++ features, notably a few functions which pass by reference (arguments with `&` in front of them). The code is presently written so as to cram as many features as possible into an **Arduino Uno**, which has only 32K of code space, 2K of working RAM, and 1K of Flash. Thus there are a lot of little oddities here and there meant to conserve space and/or speed. A few examples:

- **Division** Division on the Uno is slow and costly. Gizmo often uses custom `div` functions which are smaller and faster than a general use of the \ and `%` operators. See the file `Divison.h` Also in some cases I've found that (perhaps due to a bug?) gcc appears not to be converting division by a power of two into a simple right shift. So Gizmo often uses right shifts rather than divisions to force this. Perhaps this is cargo cult programming, perhaps not.
- **The State Machine** The state machine is a large case statement located in the go() method in `TopLevel.cpp`. To compile to a jump table, the cases have to be contiguous integers; since the Mega and Uno have different sets of cases, this means different integer `#defines`. They are NOT presently enums! Adding a new cases is always fun as you have to move all the `#defines` down or up by 1. Also, in some cases I've inlined the certain functions in the case statement as it makes the code smaller than calling them separately.
- **Helper Functions** Gizmo has lots of helper functions meant to reduce code redundancy. In several cases these helper functions have intentional side effects which you may not realize: for example, the `doMenuDisplay(...)` function [as a random example] expects to handle, and clear, the `entry` flag. These can be serious gotchas if you've not looked them over carefully.
- **Strange Code** Sometimes you'll see weird things in the code, such as code that's been inlined in the `TopLevel.cpp` state machine rather than appear in their own functions; or oddly written code which could have been written more straightforwardly; or obvious opportunities for removing redundancy through a function call. The reason behind this weirdness is almost always that the Atmel compiler for the Arduino Uno will compile tighter the way it's written.

  Indeed at some point soon I'll have to break the two codebases apart: the Uno's restrictions are making a mess of spaghetti code out of the codebase.

19.2 Code Files

The files are in certain categories:

Hardware	
`MidiShield.h/cpp`	Macros and variables for the SparkFun MIDI Shield
`LEDDisplay.h/cpp`	Code to do drawing on the Adafruit 16x8 LED
`DAC.h/cpp`	Code to control the DACs
Core Code	
`Timing.h/cpp`	Internal and external clocks, notes, pulses, beats
`TopLevel.h/cpp`	The core. Contains various important functions, the state machine.
`MidiInterface.h/cpp`	MIDI callbacks, MIDI wrapper functions
`Utility.h/cpp`	Utility functions used by various applications
`Storage.h/cpp`	Slot (file) storage in Flash
`Options.h/cpp`	Storage of global and application-specific options
`Division.h/cpp`	The arduino doesn't have hardware divide. This file contains various functions for dividing by specific common constants (10, 12, 100, etc.)
`All.h`	One header file to bring them all and in the darkness bind them.
`Gizmo.ino`	The standard Arduino entry functions
General Applications	*(Note that many parts of applications are embedded in the state machine in `TopLevel.cpp`, and a few bits are located in the MIDI callback functions in `TopLevel.cpp`, though I try to avoid embedding in the callback functions when possible.)*
`Arpeggiator.h/cpp`	The arpeggiator
`StepSequencer.h/cpp`	The step sequencer
`Control.h/cpp`	The MIDI control surface
`Gauge.h`	The MIDI gauge. There's no `Gauge.cpp` because it's entirely contained within the state machine in `TopLevel.cpp`
`Recorder.h/cpp`	The note recorder
`Split.h/cpp`	The keyboard splitter / layerer
`Thru.h/cpp`	The MIDI thru application
`Measure.h/cpp`	The Measure counter application
`Synth.h/cpp`	The Synthesizer helper top-level
Synth Helpers	*(In the `synth` subdirectory.)*
`KawaiK4.h/cpp`	NRPN→Sysex converter for the Kawai K4 and K4r
`KorgMicrosampler.h/cpp`	NRPN→Sysex converter for the Korg Microsampler
`OberheimMatrix1000.h/cpp`	CC→NRPN converter for the Oberheim Matrix 1000 with the 1.16 or 1.120 ROM upgrades
`WaldorfBlofeld.h/cpp`	NRPN→Sysex converter for the Waldorf Blofeld
`YamahaTX81Z.h/cpp`	NRPN→Sysex converter for the Yamaha TX81Z

19.3 Application Building Tutorial

Defining the Application Applications in Gizmo are included (or not) in two steps. First, you have to turn on a `#define` which will include all of the application's code. Second, you need to include the name of the application in the right spot in the application menu. We'll start with the `#define`.

We'll make an application called **Foo**. Let's assume that your application is meant to run on the Arduino Mega. In `All.h` there's an area where it says:

```
#if defined(__MEGA__)
```

```
...
#endif
```

This area defines which applications will be turned on when we compile for the Mega. In this region, add your own application:

```
#define INCLUDE_FOO
```

While we're add it, let's add a comment which lets developers know that your application is an option. You'll notice that in the file `All.h`, there is a section labeled

```
// -- APPLICATIONS --
```

Under this section, let's add a bit of documentation about our new `#define`

```
// INCLUDE_FOO                // This will include the Foo application
```

Modifying the State Machine Next we want to add our application as a state in the state machine. The state machine is for the moment a large enum called `_State`, located in the file `TopLevel.h`. The first state is state number 255: `STATE_NONE`. After this the **root state**, defined as state number 0: `STATE_ROOT`. The root state is the state that the Gizmo starts up in. The root state shows the application menu.

After the root state are up to 12 **application states**, the last of which is `STATE_OPTIONS`. These states have no associated number, and so will be numbered consecutively after 0 (the root). You will add a state here. For simplicity, add your state as the last application before `STATE_OPTIONS`. In file `TopLevel.h`, just before the line

```
    STATE_OPTIONS,
```

add the following:

```
#ifdef INCLUDE_FOO
    STATE_FOO,
#endif
```

Now that you have a state, you can add the code for it. In the file `TopLevel.cpp`, just before the line

```
        case STATE_OPTIONS:
```

add the following:

```
#ifdef INCLUDE_FOO
        case STATE_FOO:
            {
            stateFoo();
            }
        break;
#endif
```

Modifying the Root Menu We also need to modify the root menu to include your application. In the file `TopLevel.cpp`, look for the line

```
        case STATE_ROOT:
```

Here you'll find a chunk of code which looks something like this (the exact number of apps may be different):

```
#if defined(__MEGA__)
            const char* menuItems[10] = { PSTR("ARPEGGIATOR"), PSTR("STEP SEQUENCER"),
                        PSTR("RECORDER"), PSTR("GAUGE"), PSTR("CONTROLLER"), PSTR("SPLIT"),
                        PSTR("THRU"), PSTR("SYNTH"), PSTR("MEASURE"), options_p };
            doMenuDisplay(menuItems, 10, FIRST_APPLICATION, STATE_ROOT, 1);
#endif
```

You need to modify this to add a label in the menu corresponding to the position of your application in the state machine. Your application appears immediately before the Options application, so you'd modify it this way:

```
#if defined(__MEGA__)
            const char* menuItems[11] = { PSTR("ARPEGGIATOR"), PSTR("STEP SEQUENCER"),
                        PSTR("RECORDER"), PSTR("GAUGE"), PSTR("CONTROLLER"), PSTR("SPLIT"),
                        PSTR("THRU"), PSTR("SYNTH"), PSTR("MEASURE"), PSTR("FOO"), options_p };
            doMenuDisplay(menuItems, 11, FIRST_APPLICATION, STATE_ROOT, 1);
#endif
```

Notice h

This adds a state which, when called, will call the `stateFoo()` method. Additional states would typically get tacked onto the end of the `#define` list and be called stuff like `STATE_FOO_EDIT` or `STATE_FOO_WHATEVER`. The critical item here is that your `#defines` in `TopLevel.h` and `TopLevel.cpp` must be in the same position, so the switch statement stays contiguous and can remain a jump table.

Now, create some `Foo.h` and `Foo.cpp` files, and add `Foo.h` to `All.h` as an `#include`.

Your `Foo.h` file should look like this:

```
#ifndef __FOO_H__
#define __FOO_H__
#ifdef INCLUDE_FOO

void stateFoo();

#endif
#endif
```

Your `Foo.cpp` file should look like this:

```
#include "All.h"
#ifdef INCLUDE_FOO

void stateFoo()
    {
    }

#endif
```