Audio on/off switch: ezdac~

In this tutorial patch, the **dac~** object which was used in earlier examples has been replaced by a button with a speaker icon. This is the **ezdac~** object, a user interface object available in the object palette.

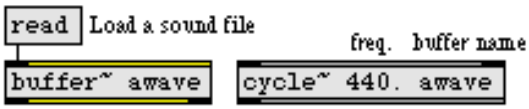
ezdac~ is an on/off button for audio, available in the object palette

The **ezdac~** works much like **dac~**, except that clicking on it turns the audio on or off. It can also respond to start and stop messages in its left inlet, like **dac~**. (Unlike **dac~**, however, it is appropriate only for output channels 1 and 2.) The **ezdac~** button is highlighted when audio is on.

A stored sound: buffer~

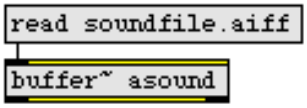
In the previous examples, the **cycle~** object was used to read repeatedly through 512 values describing a cycle of a cosine wave. In fact, though, **cycle~** can read through any 512 values, treating them as a single cycle of a waveform. These 512 numbers must be stored in an object called **buffer~**. (A *buffer* means a holding place for data.)

A **buffer~** object requires a unique name typed in as an argument. A **cycle~** object can then be made to read from that buffer by typing the same name in as its argument. (The initial frequency value for **cycle~**, just before the buffer name, is optional.)



cycle~ reads its waveform from a buffer~ of the same name

To get the sound into the **buffer~**, send it a read message. That opens an Open Document dialog box, allowing you to select an audio file to load. The word read can optionally be followed by a specific file name, to read a file in without selecting it from the dialog box, provided that the audio file is in Max's search path.



Read in a specific sound immediately

Regardless of the length of the sound in the **buffer~**, **cycle~** uses only 512 samples from it for its waveform. (You can specify a starting point in the **buffer~** for **cycle~** to begin its waveform, either with an additional argument to **cycle~** or with a set message to **cycle~**.) In the example patch, we use an audio file that contains exactly 512 samples.

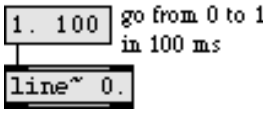
> **Technical detail:** In fact, **cycle~** uses 513 samples. The 513th sample is used only for interpolation from the 512th sample. When **cycle~** is being used to create a periodic waveform, as in this example patch, the 513th sample should be the same as the 1st sample. If the **buffer~** contains only 512 samples, as in this example, **cycle~** supplies a 513th sample that is the same as the 1st sample.

- Click on the **message** box that says read gtr512.aiff. This loads in the audio file. Then click on the **ezdac~** object to turn the audio on. (There will be no sound at first. Can you explain why?) Next, click on the **message** box marked B3 to listen to 1 second of the **cycle~** object.

There are several other objects that can use the data in a **buffer~**, as you will see in later chapters.

Create a breakpoint line segment function with line~

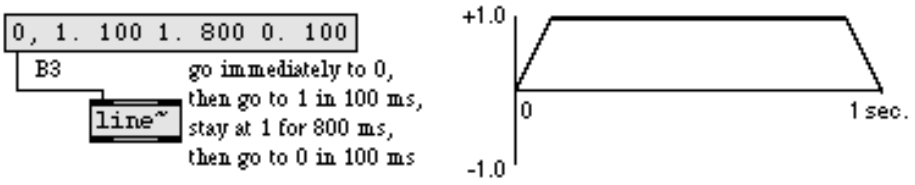
In the previous example patch, we used **line~** to make a linearly changing signal by sending it a list of two numbers. The first number in the list was a target value and the second was the amount of time, in milliseconds, for **line~** to arrive at the target value.



line~ is given a target value (1.) and an amount of time to get there (100 ms)

If we want to, we can send **line~** a longer list containing many value-time pairs of numbers (up to 64 pairs of numbers). In this way, we can make **line~** perform a more elaborate function composed of many adjoining line segments. After completing the first

line segment, **line~** proceeds immediately toward the next target value in the list, taking the specified amount of time to get there.

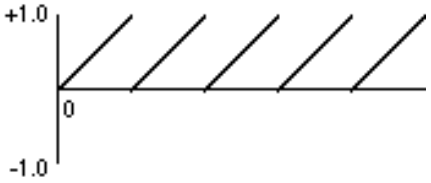


A function made up of line segments

Synthesizer users are familiar with using this type of function to generate an "ADSR" amplitude envelope. That is what we're doing in this example patch, although we can choose how many line segments we wish to use for the envelope.

Other signal generators: phasor~ and noise~

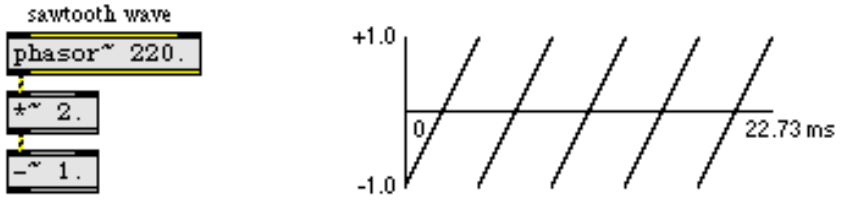
The **phasor~** object produces a signal that ramps repeatedly from 0 to 1.



Signal produced by phasor~

The frequency with which it repeats this ramp can be specified as an argument or can be provided in the left inlet, in Hertz, just as with **cycle~**. This type of function is useful at sub-audio frequencies to generate periodically recurring events (a crescendo, a filter sweep, etc.). At a sufficiently high frequency, of course, it is audible as a sawtooth waveform.

In the example patch, the **phasor~** is pitched an octave above **cycle~**, and its output is scaled and offset so that it ramps from -1 to +1.



220 Hz sawtooth wave

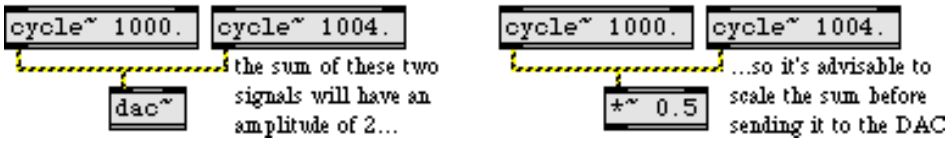
Technical detail: A sawtooth waveform produces a harmonic spectrum, with the amplitude of each harmonic inversely proportional to the harmonic number. Thus, if the waveform has amplitude A, the fundamental (first harmonic) has amplitude A, the second harmonic has amplitude A/ 2, the third harmonic has amplitude A/3, etc.

The **noise~** object produces white noise: a signal that consists of a completely random stream of samples. In this example patch, it is used to add a short burst of noise to the attack of a composite sound.

- Click on the **message** box marked B1 to hear white noise. Click on the **message** box marked B2 to hear a sawtooth wave.

Add signals to produce a composite sound

Any time two or more signals are connected to the same signal inlet, those signals are added together and their sum is used by the receiving object.

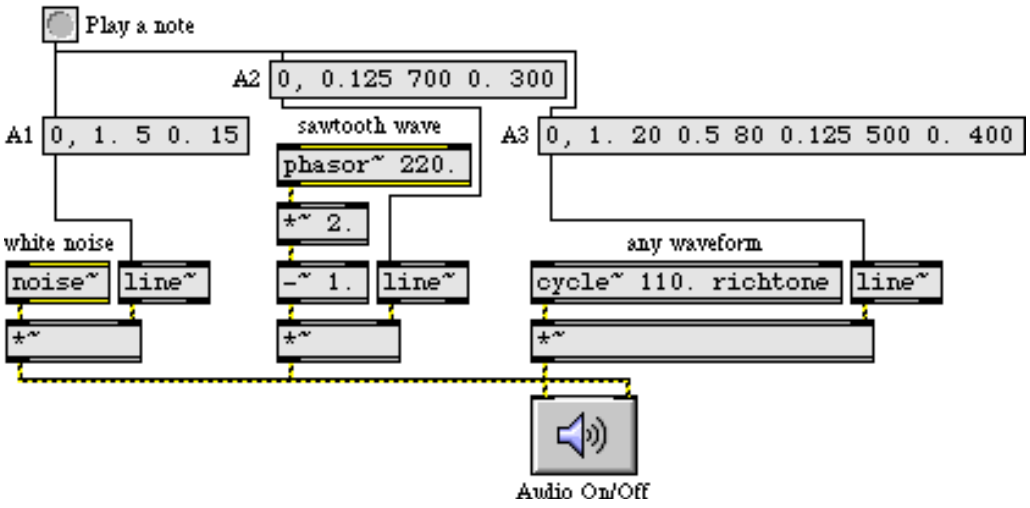


Multiple signals are added (mixed) in a signal inlet

Addition of digital signals is equivalent to unity gain mixing in analog audio. It is important to note that even if all your signals have amplitude less than or equal to 1, the sum of such signals can easily exceed 1. In MSP it's fine to have a signal with an

amplitude that exceeds 1, but before sending the signal to **dac~** you must scale it (usually with a ***~** object) to keep its amplitude less than or equal to 1. A signal with amplitude greater than 1 will be distorted by **dac~**.

In the example patch, white noise, a 220 Hz sawtooth wave, and a 110 Hz tone using the waveform in **buffer~** are all mixed together to produce a composite instrument sound.



Three signals mixed to make a composite instrument sound

Each of the three tones has a different amplitude envelope, causing the timbre of the note to evolve over the course of its 1-second duration. The three tones combine to form a note that begins with noise, quickly becomes electric-guitar-like, and gets a boost in its overtones from the sawtooth wave toward the end. Even though the three signals crossfade, their amplitudes are such that there is no possibility of clipping (except, possibly, in the very earliest milliseconds of the note, which are very noisy anyway).

- Click on the **button** to play all three signals simultaneously. To hear each of the individual parts that comprise the note, click on the **message** boxes marked *A1*, *A2*, and *A3*. If you want to hear how each of the three signals sound sustained at full volume, click on the **message** boxes marked *B1*, *B2*, and *B3*. When you have finished, click on **ezdac~** to turn the audio off.

Summary

The **ezdac~** object is a button for switching the audio on and off. The **buffer~** object stores a sound. You can load an audio file into **buffer~** with a read message, which opens an Open Document dialog box for choosing the file to load in. If a **cycle~** object has a typed-in

argument which gives it the same name as a **buffer~** object has, the **cycle~** will use 512 samples from that buffered sound as its waveform, instead of the default cosine wave.

The **phasor~** object generates a signal that increases linearly from 0 to 1. This ramp from 0 to 1 can be generated repeatedly at a specific frequency to produce a sawtooth wave. For generating white noise, the **noise~** object sends out a signal consisting of random samples.

Whenever you connect more than one signal to a given signal inlet, the receiving object adds those signals together and uses the sum as its input in that inlet. Exercise care when mixing (adding) audio signals, to avoid distortion caused by sending a signal with amplitude greater than 1 to the DAC; signals must be kept in the range -1 to +1 when sent to **dac~** or **ezdac~**.

The **line~** object can receive a list in its left inlet that consists of up to 64 pairs of numbers representing target values and transition times. It will produce a signal that changes linearly from one target value to another in the specified amounts of time. This can be used to make a function of line segments describing any shape desired, which is particularly useful as a control signal for amplitude envelopes. You can achieve crossfades between signals by using different amplitude envelopes from different **line~** objects.

See Also

buffer~	Store audio samples
ezdac~	Audio output and on/off button
phasor~	Sawtooth wave generator
noise~	White noise generator