MZ C Wave Processor: Reference Guide

Author: Matt Zapp, 12/05/2014

Table of Contents

1. INTRODUCTION

Welcome! The following documentation should help you get started using this command-line audio processor that I have not-so-creatively dubbed MZ C Wave Processor. The program accepts simple 16-bit *.wav files, and takes its commands from a JSON-formatted text file detailing the kinds of processing that you would like to perform on the file resulting in an output file with the selected effects applied. All processing is done internally with floating-point precision to allow for greater accuracy and better sound quality, but the eventual file will be saved with the same 16-bit format.

The line required to run this application is:
./MZCWaveProcessor <input file>.wav <output file name>.wav <JSON-formatted text file>

2. HOW TO INSTALL

The application should run without any difficulty on modern OS X machines. I did try to make the code itself as platform agnostic as possible, but the wave format header requires some awareness of its host system's endianness and thus required a few system-specific function calls that prevent this code from being compiled across platforms. Future versions of this application will be built to compile and run on all systems.

If building the application from code, you will need the following:

1. The C/C++ version of Eclipse IDE for OS X (Kepler).
    a. This also may require Java for OS X
2. The Eclipse project zip file.

To build the application:

1. Download the MZCWaveProcessor.zip archive file.
2. Open Eclipse
3. Go to the workbench
4. Select "Import" from the file menu
5. When prompted, select General/Existing Projects into Workspace and hit next
6. Use the "Select Archive File" option, and click "Browse…"
7. Open the MZCWaveProcessor.zip from the file system wherever you may have downloaded it
8. This should display "Final Project" as the available internal project. Click "Finish" to import the project.
9. Clicking the little hammer in the toolbar or selecting "Build Project" from the Project menu will build the application and place it in the "Default" folder.

3. PROCESSING YOUR AUDIO

A. Supported input *.WAV formats

The MZ C Wave Processor should support most 16-bit stereo *.wav files formatted in the "canonical" form of RIFF. A few demonstration *.wav files are included with the source code, but feel free to try others as well. The RIFF format has been around for a long time and people have built upon it substantially so many files (especially those with bit-depths larger than 16) will likely fail to load due to extra data contained in the file's header. This extra information can include metadata, extra padding, extra channels, playlists, and others, all of which will cause the program to exit prematurely with an error to give you a clue as to what part of the header contained the extra information.

B. JSON Process-File syntax

The MZ C Wave Processor takes a command file to eliminate the need for complex strings of command-line arguments. This allows you to sequence a number of processes without ever having to leave 32-bit floating-point processing land, and allows for compensation of processes which may increase the gain to the point of what would be clipping in a 16-bit file. This file uses JavaScript Object Notation (JSON) to parse its commands, and is relatively flexible. Not all parameters need to be specified. If a process's parameters are not all available, or are misspelled, a program default is substituted instead.

When writing a JSON file for the MZ C Wav Processor, the format will be, conceptually, as follows written in all lower-case. Explanatory comments are shows with "//" and values to be replaced with specific names and values are show with "<>". Both should be omitted in an actual processing file:

```
//the whole file must be wrapped in curly-braces so that the program recognizes it as an object
{
        // The "process" tag followed by  a colon and an opening curly-brace signifies a process
        "process": {
                // Each process object must have one type
                "type": "<type of process goes here>",
                // Each process object needs one params object if not using defaults.
                "params": {
                        "<parameter name>": <parameter value as a floating-point number>,
                        "<parameter name>": <parameter value as a floating-point number>
                }
        }
}
```

C. Processes and Parameters

Reverse: Reverses the playback direction of a file and takes no parameters.
Gain: This changes the over-all volume of a file

- ***mult:*** The multiplication factor of gain. Numbers less than 1 will reduce the volume, numbers greater than one will increase it.

Stretch: Stretches or compresses a file in length by playing back through it faster or slower, similar to what happens when you change the playback rate of a tape. This process will produce "aliasing."

- ***ratio:*** The ratio by which to stretch the file. Numbers less than 1 will increase the file length, numbers higher than 1 will shorten it.

Distort: Clips or changes the linearity of a file. This process will produce "aliasing."

- ***algorithm:*** The style by which distortion is applied. 0 is "soft" 1 is "hard."
- ***knee:*** the point at which the distortion kicks in. Values range from 0 (immediately) to 32767 (never on a 16-bit file) and higher.

Delay: Shifts a file in time by an arbitrary number of samples.

- ***samples:*** The number of samples to shift the audio. Values can be nearly any floating-point number. Wave files do have a length limit so don't expect huge values to work particularly well.

Echo: Repeats your file over and over again with some amount of decay. Like yelling into a canyon.

- ***samples:*** The number of samples between each echo. Can, again, be almost any floating-point value, but keep in mind the limits of reality/your patience.
- ***feedback:*** The amount of each echo that gets fed back into the echo processes. Value ranges between 0 and slightly less than 1. Keep in mind that values very close to 1 will cause the processing time to carry on for a VERY long time and may over-flow the max size of a wave file.

Tremolo: Modulates the file's amplitude over time with an on-board oscillator.

- ***shape:*** The shape of the modulating wave. Wave shapes are listed later.
- ***freq, frequency, or rate:*** The frequency of the modulating wave. This isn't bounded by anything, but keep in mind this oscillator is also subject to Nyqist and will give you bizarre values if you try to specify something over half of the sampling-rate of your input file.
- ***depth:*** How much the wave will affect the amplitude of the signal. Values of 0 – 1 will go from 0% to 100% modulation, values > 1 will start increasing the volume of your file and remain at 100% depth.

Ringmod: Is identical to Tremolo, except the modulating wave is bi-polar and will invert the signal. This creates some really wild robot noises at higher frequencies.
Vibrato: Applies a variation to the pitch of a file. This process will produce "aliasing."

- ***shape:*** The shape of the modulating wave. Wave shapes are listed later.
- ***freq, frequency, or rate:*** The frequency of the modulating wave. This isn't bounded by anything, but keep in mind this oscillator is also subject to Nyqist and will give you bizarre values if you try to specify something over half of the sampling-rate of your input file.

- ***depth:*** How much the wave will affect the pitch of the signal. Values of 0 – 1 will go from 0% to 100% modulation, values > 1 will start folding the signal in strange and unexpected ways.

Flange: An interesting mix of echo and vibrato. This process will produce "aliasing."

- ***shape:*** The shape of the modulating wave. Wave shapes are listed later.
- ***freq, frequency, or rate:*** The frequency of the modulating wave. This isn't bounded by anything, but keep in mind this oscillator is also subject to Nyqist and will give you bizarre values if you try to specify something over half of the sampling-rate of your input file.
- ***depth:*** How much the wave will affect the pitch of the signal. Values of 0 – 1 will go from 0% to 100% modulation, values > 1 will start introducing clicks and pops as the read and write heads of the echo start to skip and jump over each other.
- ***feedback:*** how much of the output is fed back into the flanger. More will create a very dramatic swooshing, less will be far more subtle. Value ranges between 0 and slightly less than 1. The delay times in the Flanger are short enough to avoid ludicrously long processing times at very high feedback rates.

Lowpass: PROTOTYPE: This *should* filter off the high frequencies and apply resonance. However, the DSP is massively unstable at this point so please use at your own risk. 9 times out of 10 it will work ok-ish if Q is kept fairly low (< 0) but it is prone to exploding and creating all DC offset, or one obscenely loud, high frequency, both of which are not ideal.

- ***cutoff:*** The frequency at which the gain-reduction will start to be applied.
- ***q:*** The "quality" of the filter. High Qs blow up the filter, but should apply resonance.