The Artisan Sound Engine

Using the Artisan Sound Engine

November 15, 2009

Artisan Instruments, Inc.
6450 NE 183
Kenmore, WA 98028
425-486-6555
artisan@interserv.com
sales@artisan-instruments.com
emarka@mac.com

Table of Contents

1 Installing the Sound Engine

Normally the sound engine will come to you already installed by an Artisan technician. In those rare cases where it is to be installed on site by the end user or customer, special instructions will need to be followed. Please request instructions directly from Artisan. The sound engine is only authorized to run on Ubuntu Linux and is tied securely directly to the computer hardware upon which it will run. The computer running the sound engine must be totally dedicated to that purpose and no other programs or uses are authorized under the Artisan warranty. The sound engine will not run without obtaining an authorized key code file directly from Artisan under any circumstances. The Artisan Sound Engine is designed to play Artisan Digital ARNK files, but Artisan may also agree to authorize your sound engine to play WAV and AIFF files if the need arises. Contact Artisan for a new key code file if this feature is desired.

2 Writing the Configuration File

Make sure to create a plain text file when you output from your editor.

2.1 Expressions

Throughout the rest of this document, you will find references to expressions. In most cases, this will simply be the name of a bit or a controller. However, much flexibility is made available by writing more complicated expressions. You may wish to skim this section, and then return for a more careful reading when it's more clear what expressions are good for.
All expressions are enclosed in (parentheses) to set them off from other syntactic elements.

2.1.1 Constants

A constant is simply a number, such as 10 or -5.

2.1.2 Notes

A specific MIDI note can be referenced in an expression by giving its MIDI channel and note number. For example, note 23 on channel 2 would be “c2n23”. The channel number can be from 1 to 16, and the note number from 0 to 127. There may not be embedded spaces, and the letters may be either upper or lower case.
The value of a note is zero when the note is off, and non-zero when the note is on. The actual value will be the velocity which was received with the note on message.

2.1.3 Program Change messages

Program change messages are not valid in expressions.

2.1.4 Controllers

A controller can be referenced in an expression by giving its MIDI channel and controller number. For example, controller 17 on channel 16 would be "C16C17". The channel number can be from 1 to 16, and the controller number from 0 to 127. There may not be any embedded spaces, and the letters may be either upper or lower case.
The value of a controller is the value most recently received. This value will be between 0 and 127 inclusive. If no value has ever been received for a controller, its value in an expression will be -1.

2.1.5 Arithmetic

Arithmetic can be performed in an expression.

2.1.5.1 Addition: +

The expressions to the left and right of the + are added.

2.1.5.2 Subtraction: -

The expression to the right of the – is subtracted from the one to the left.

*2.1.5.3 Multiplication: **

The expressions to the left and right of the * are multiplied.

2.1.5.4 Division: /

The expression to the left of the / is divided by the expression to the right. The result is truncated to an integer.

2.1.5.5 Modulo: %

The expression to the left of the % is divided by the expression to the right, and the remainder of this division is returned.

2.1.5.6 Not: !

If the expression to the right of the ! is zero, one is returned. If the expression is non-zero, zero is returned.

2.1.5.7 Complement: ~

The 32-bit complement of the number to the right of the ~ is returned.

2.1.5.8 Shift Left: <<

The number to the left of the << is shifted left the number of binary places as the expression to the right of the <<. x << 1 multiplies x by two, and x << 2 multiplies by four. In most cases, multiplication is a better choice.

2.1.5.9 Shift Right: >>

The number to the left of the >> is shifted right the number of binary places as the expression to the right of the >>. x >> 1 divides x by two, and x >> 2 divides by four. In most cases, division is a better choice.

2.1.5.10 Less Than: <

If the value of the expression to the left of < is less than the value to the right, one is returned, otherwise zero is returned.

2.1.5.11 Less Than or Equal To: <=

If the value of the expression to the left of <= is less than or equal to the value to the right, one is returned, otherwise zero is returned.

2.1.5.12 Greater Than or Equal To: >=

If the value of the expression to the left of >= is greater than or equal to the value to the right, one is returned, otherwise zero is returned.

2.1.5.13 Greater Than: >

If the value of the expression to the left of > is greater than the value to the right, one is returned, otherwise zero is returned.

2.1.5.14 Equal To: = =

If the value of the expression to the left of = = is equal to the value to the right, one is returned, otherwise zero is returned. In this document I have put a space between the two equal signs to make it clear that “equal to” is actually two equal signs. If I didn't, they would look like one long equal sign with some fonts. The space must not be present in the configuration file.

2.1.5.15 Not Equal To: !=

If the value of the expression to the left of != is not equal to the value to the right, one is returned, otherwise zero is returned.

2.1.5.16 Binary And: &

The binary AND of the expressions to the left and right of & is returned.

2.1.5.17 Binary Or: |

The binary OR of the expressions to the left and right of the | is returned.

2.1.5.18 Exclusive Or: ^

The binary exclusive OR of the expressions to the left and right of the ^ is returned.

2.1.5.19 Parentheses: ( )

Portions of expressions may be grouped to indicate order of computation. (2 * 3 + 4) is 10, but (2 * (3 + 4)) is 14.

2.2 Inputs: Divisions, bits, and controllers

2.2.1 *DIVISION

A division statement defines how notes get into the sound engine. The template for a *division statement is
*division <name> <footage> <count> <midi note>
<name> is the name of the division, which will be referenced later in the file.
<footage> is the default footage of the division, normally just 8.
<count> is the number of notes in the division, usually 61.
<midi note> is the MIDI note of the lowest note of the division, for example C1N36.
A typical statement might be
*division solo 8 61 c1n36
or for a pedalboard
*division pedal 16 25 c1n24

2.2.2 *BIT

A bit statement is used to attach a name to one particular MIDI note. This is usually used to define stop tabs. The template for a bit statement is
*bit <name> <midi note>
For example
*bit solo_flute_8 c12n55
This specifies that the name "solo_flute_8" will refer to the current state of note 55 in MIDI channel 12.

2.2.3 *VALUE

The value statement allows a name to be associated with an expression. The template is
*value <name> (<expression>)
For example,
*value transpose (c1c7 – 6)
This takes the current value of controller 7 on channel 1 and subtracts six. This number is then substituted later in the file wherever the name "transpose" is used.

2.3 Couplers

Couplers may be specified as desired. In fact, couplers MUST be specified, or there will be no coupling, and no sound generated!

2.3.1 *COUPLER

The template for a normal coupler is
*coupler (<expression>) <from_division> to <to_division> <footage>
Here, the <expression> tells when the coupler is engaged. If the coupler is to be always on, just use "(1)".
The <from_division> is the division from which the stops are taken.
"to" is the word to, which must be written just as shown.
The <to_division> is the division which supplies the note information.
<footage> specifies whether an offset is to be made; use "8" for a unity coupler. Use 16 for a sub-octave, or 4 for an octave coupler.
All divisions which don't have a unison off tab must have an unconditional unison coupler such as
*coupler (1) swell to swell 8
To implement unison off, use something like
*coupler (!swell_unison_off) swell to swell 8
and an octave coupler might look like
*coupler (swell_octave) swell to swell 4

2.3.2 *BASS

A bass coupler is just like a normal coupler in function and syntax, with two differences: Substitute the word *bass for *coupler in the definition file, and the coupler will only couple the lowest note being played on the division at any time.

2.3.3 *MELODY

A melody coupler is just like a normal coupler in function and syntax, with two differences: The word *melody is substituted for *coupler in the definition file, and the coupler will only couple the highest note being played on the division at any time.

2.4 Ranks, Traps, and other outputs

Ranks are collections of notes, specified either as an Artisan rank file or as individual .wav or .aiff files. The ability to play .wav and .aiff files is an option. Traps are kind of like one-note ranks, and the sound file is usually a sound effect instead of a pipe sound.

2.4.1 *RANK

The template of a rank looks like
*rank <name> <footage> <filename(s)>
Here, <name> is the name of the rank, to be referenced later on.
<footage> specifies the pitch of the lowest note of the rank. If a non-integer footage is needed, it is specified in a form such as "1 – 5 / 8". There must be a space between the – and the 5, or the silly parser thinks it's the number -5.
The <filename(s)> is usually the path to the Artisan rank file, which usually has a name

which ends in ".arnk". If you have the optional .wav and/or .aiff capability, then <filename(s)> must expand to all the .wav or .aiff files which make up the rank, in order from low to high. This is done by using a wild card, and by careful naming of the files. The system sorts the files in alphabetical order, so if the note number (perhaps with leading zeros) is in the file name, they will end up in the correct order.
A sample rank line:
*rank tenor_sax 8 "ranks/tenor_sax.arnk"
An example with .wav files:
rank tibia 16 "ranks/tibia/.wav"
Ranks may have a number of modifiers. These should be after the *rank definition.

2.4.1.1 *CHANNEL

*channel specifies that the input information is to be routed to one particular channel. If the input data is in stereo, the two channels will be added together before being routed to the selected channel. *channel 1 specifies the left channel, and *channel 2 specifies the right channel.

2.4.1.2 *STEREO

*stereo specifies that the rank is to go out to both channels. If the rank is recorded in stereo, the stereo information will be maintained. If the rank is mono, the same signal will be routed to both channels. This statement is always written
*stereo 1

2.4.1.3 *LEVEL

*level allows the relative sound intensity level of the entire rank to be adjusted. The template is
*level <relative level>
Here, <relative level> is a number which specifies how much the rank is to be amplified or attenuated. The units are dB, and may be a floating point number. A number of zero means no amplification or attenuation. A positive number means amplify, and a negative number means attenuate. Each multiple of three means an approximate doubling of the sound power. Thus, specifying *level 6.0 means that the sound should be four times as loud, and *level -9.0 means the sound should be 1/8 as loud. The maximum value of a positive level is limited by the maximum values of the samples in the rank. The rank values may not exceed +/- 32767 after being modified by the *level statement; if they are, a warning message is displayed and the level is modified. This occurs independently for each note in the rank, so several such messages could occur for one rank.

2.4.1.4 *NOTELEVEL

*notelevel is the same as *level, except that it applies to one note only. The template is

*notelevel <note number> <relative level>
Here, <note number> is the number of the note to modify, with 1 being the lowest note in the rank. <relative level> is the number of dB the note is to be adjusted. This adjustment is on top of any adjustment which may have been specified with *level.

*2.4.1.5 *TREM*

*trem specifies that an synthetic tremulant should be applied to this rank. The template is
*trem <trem_index>
<trem_index> specifies which tremulant should be connected to this rank. See *tremdef later in this document for how to define a tremulant.

*2.4.1.6 *RELEASE*

*release modifies the way in which a note turns off. The template is
*release <time in mSec>
Normally, if a rank has samples of the sound of a note dying out, the sound engine uses these samples when the note is to stop playing. Sometimes this is not desirable, for example when the note was recorded in a very reverberant space, and this reverberation is not desired. The *release command causes the note to be faded out over the period of time specified by <time in mSec>. An example might be
*release 200

*2.4.1.7 *NOLOOP*

Sometimes it is desired to play a sampled note or sound straight through without playing a sustain loop while the note is held. The *noloop keyword tells the sound engine to ignore any looping information in the rank, and play straight through. It is written without any modifiers:
*noloop

*2.4.1.8 *DAMPER*

For some sounds, it may be desirable to have a controllable damper. When the damper is engaged (as in the normal "up" position of a piano damper pedal), the sound will die out rapidly when the key is released. When the damper is not engaged (the depressed position of a piano pedal), the sound will play normally, either in its entirety if there is no release section marked, or until the key is released, if a release section is marked.
The template for the *damper statement is
*damper (<damper_enable>) <damper_time>
<damper_enable> is an expression which controls when the damper should be engaged. If it evaluates to a non-zero value, the damper is engaged, otherwise the damper is released.
<damper_time> is an integer which specifies how quickly the damper should act.

This is the number of milliseconds that it takes for the sound to fade by 60 dB.
An example damper statement might be
*damper (!piano_damper) 400
Now, since in a normal instrument like a piano the "damper" is normally engaged and you depress the pedal to actually disengage it, the expected statement for using the *DAMPER would be with the !(not) preceding the <damper_enable> boolean.

2.4.2 *TRAP_FILE

The trap_file statement is a lot like a *rank statement, except it allows a single sound to be associated with a name. It is normally used for traps, where the same sound is commonly played for any note within the division. The template is
*trap_file <name> "<file_name>"
Here, <name> is the name to be associated with the trap, and <file_name> is the name of the file which contains the trap sound. For example,
*trap_file chinese_gong "traps/chinese_gong.arnk"
The *trap_file may be modified by one or more of the following commands.

*2.4.2.1 *CHANNEL*

*channel specifies that the input information is to be routed to one particular channel. If the input data is in stereo, the two channels will be added together before being routed to the selected channel. *channel 1 specifies the left channel, and *channel 2 specifies the right channel.

*2.4.2.2 *STEREO*

*stereo specifies that the trap is to go out to both channels. If the trap is recorded in stereo, the stereo information will be maintained. If the trap is mono, the same signal will be routed to both channels. This statement is always written
*stereo 1

*2.4.2.3 *LEVEL*

*level allows the relative sound intensity level of the trap to be adjusted. The template is
*level <relative level>
Here, <relative level> is a number which specifies how much the trap is to be amplified or attenuated. The units are dB, and may be a floating point number. A number of zero means no amplification or attenuation. A positive number means amplify, and a negative number means attenuate. Each multiple of three means an approximate doubling of the sound power. Thus, specifying *level 6.0 means that the sound should be four times as loud, and *level -9.0 means the sound should be 1/8 as loud. The maximum value of a positive level is limited by the maximum values of the samples in the trap. The trap values may not exceed +/- 32767 after being modified by the *level statement; if they are, a warning message is displayed and the

level is modified.

2.4.2.4 *NOLOOP

Sometimes it is desired to play a sound straight through without playing a sustain loop while the note is held. The *noloop keyword tells the sound engine to ignore any looping information in the trap, and play straight through. It is written without any modifiers:
*noloop

2.5 Stops: Connecting inputs to outputs

2.5.1 *STOP

The *stop statement allows a rank to be connected to a division, usually when a particular input is enabled. The template for the *stop statement is
*stop (<expression>) <division> <rank> <footage>
or the optional form
*stop (<expression>) <division> <rank> <footage> <first_note> <last_note>
<expression> should be non-zero when the stop is to be active. It is usually just the name of a tab.
<division> is the name of a previously defined division that is to supply the note information to play the rank.
<rank> is the name of a previously defined rank which is to produce the sound.
<footage> specifies a possible modification of the pitch of the rank. 8 means no modification. 16 is down one octave, and 4 is up one octave.
If supplied, <first_note> limits which notes may be played by this stop. For example, if <first_note> is 3, the first two notes of the rank would not sound.
<last_note> limits the notes played at the high end of the rank. If <last_note> is less than the number of notes in the rank, then some of the top notes of the rank will not sound.
Stops may be modified by one of the following two statements:

2.5.1.1 *PIZZ

The *pizz statement specifies that the stop should be pizzicato. The template is
*pizz <low_time> <high_time>
<low_time> is the number of milliseconds that the bottom note of the rank should play. <high_time> is the number of milliseconds that the top note of the rank should play. Notes in between are spaced evenly between. An example might be:
*pizz 300 200

2.5.1.2 *REIT

The *reit statement specifies that the stop should reiterate. The template is
*reit <low_rate> <high_rate>

<low_rate> specifies how many times per second the lowest note of the rank should play, and <high_rate> specifies how many times per second the highest note of the rank should play. Notes in between are spaced evenly between these values. An example might be:
*reit 8 15

2.5.2 *TRAP_STOP

A trap_stop is just like a stop, except that it causes a trap to sound instead of notes in a rank. The template is
*trap_stop (<expression>) <division> <trap>
Here, <division> is the name of the previously defined division which plays the trap. <trap> is the name of the previously defined trap_file.
Normally, any time any note in the division is played, the trap will sound once. The behavior may be modified with one or more of the following:

*2.5.2.1 *DOWNBEAT*

The trap can be set to sound only when a selected number of notes are played on the division. The template is
*downbeat <note_count>
Here, <note_count> is the number of notes which must be held for the trap to sound. The most common case would be
*downbeat 1
This causes the trap to sound only as the first note is played on the division. Playing a second note while holding the first does not sound the trap. To sound the trap again, all notes must be released, and then it will sound when a note is played again.

*2.5.2.2 *BACKBEAT*

*backbeat causes the trap to sound when the last note on a division is released. It is written simply
*backbeat
If a chord is played, the trap will not sound until the last note of the chord is released.

*2.5.2.3 *PIZZ*

*pizz causes the trap to sound for only the length of time specified. The template is
*pizz <time>
For example,
*pizz 300
would cause the trap to sound for only 300 mSec.

*2.5.2.4 *REIT*

*reit causes the trap to repeat as long as a note is held. The template is

*reit <rate>
Here, <rate> specifies how many times per second the trap is to repeat. An example might be
*reit 5
This would cause the trap to repeat five times per second.

2.6 Directives

This section describes commands which are global in scope.

2.6.1 *SHADES

Expression is applied on a channel basis. The *shades command specifies the desired expression for one of the channels. The template is
*shades <channel> (<expression>)
Here, <channel> is 1 for the left channel, and 2 for the right channel.
<expression> specifies how the level of the channel is to be modified, and is written in dB. Thus, a value of 0 does not modify the level, a negative number attenuates the level, and a positive number amplifies the level.
A controller is usually used to set the expression level. A little extra care is usually taken in writing this expression so that a reasonable level is applied if no value has been received for the selected controller. Controllers have a value of -1 (which is otherwise impossible) until a valid controller message is received for that controller. This lends itself to a *shades command of the following sort:
*shades 1 ((swell_shoe != -1) * (swell_shoe / 3 – 30))
Let's go over that in detail. First, we're assuming that swell_shoe is the name of a controller (from a *value statement) which is to adjust the level of the left channel (channel 1). We're assuming that it will take on the values of 0 to 90, and the total desired range of attenuation is 30 dB. The (swell_shoe != -1) part of the expression will evaluate to 0 if a value has never been received for swell_shoe, and the sound level will not be modified. When swell_shoe is not -1, that subexpression will evaluate to 1, and will allow the second part of the expression to influence the final value. The controller values of 0 through 90 will now cause attenuation of 0 through -30 dB.

2.6.2 *TREMDEF

The *tremdef statement is used to create a tremulant, which can then be connected to ranks as desired using the *trem modifier. The template is
*tremdef <trem_index> (<expression>) <speed> <depth>
Here, <trem_index> is an arbitrary index number, which allows this *tremdef to be connected to a *trem statement in a rank.
<expression> tells when the tremulant should be on.
<speed> is a floating point number which tells how many times per second the tremulant should beat. Valid numbers are from 1.0 to 20.0.
<depth> specifies how strong the tremulant should be, from 0.0 to 100.0.

An example might be
*tremdef 1 (main_trem_on) 8.2 60.0

2.6.3 *TUNE

*tune allows the pitch of the entire sound engine to be modified. The template is
*tune (<expression>)
where <expression> specifies the relative pitch adjustment from nominal in cents. The ability to tune may be limited by the capabilities of the sound card in use.
An example *tune statement might be
*tune ((c12c17 != -1) * (c12c17 – 64))
Here, it is assumed that controller messages arriving on channel 12 controller 17 are to control the pitch of the sound engine. There is nothing special about this particular controller; the number was chosen randomly. When the controller has not been seen by the sound engine, its value will be -1, and the expression will evaluate to zero, not affecting the sound engine. Otherwise, values between 0 and 127 will cause the pitch of the sound engine to be adjusted from -64 to +63 cents, respectively.
Note that changing the tuning of the sound engine is somewhat disruptive, and care should be taken not to adjust it during performance.

2.6.4 *ANTI_CLIP

Clipping is what happens when the sum total of all the sounds being played by the organ exceed the ability of the sound card to reproduce those sounds. The result is a very unpleasant and unmusical noise. Whenever this is about to happen, the sound engine abruptly decreases its overall sound level so as to prevent this clipping. This abrupt change is volume could be disconcerting.
*anti_clip is a master gain control for the sound engine. By specifying a smaller *anti_clip, the sound level can be reduced to that point that clipping is never in danger of occurring, thus preventing the sudden level changes. The template is
*anti_clip <value>
Here, value is the linear number (not dB) which is multiplied by all samples before they are sent to the sound card. A way to find this number is to configure the sound engine, and then play it as loudly as you can. That is, turn up the expression pedals, turn on all the stops, and then play as many notes at once as you're likely to ever play. Then, on the computer, stop the sound engine (CTRL/C, for example), and it will print out the anti_clip level which was necessary in order to prevent clipping. Use this number (or a slightly smaller one) in an *anti_clip statement in your configuration file.

2.6.5 *TRANSPOSE

*transpose allows the entire sound engine to be adjusted up or down an integral number of semi-tones. Its template is
*transpose (<expression>)
<expression> should evaluate to the number of semitones that the sound is to be

modified upwards; negative numbers cause downward transposition. An example might be

*transpose ((transpose != -1) * (transpose – 6))

Assuming that transpose is the name of a controller which might assume the values of 0 through 12, here's how this statement will work: If no messages have arrived on the transpose controller, its value will be -1, the expression will evaluate to zero, and no transpose will occur. When values do arrive, values of 0 through 12 will cause the expression to evalute to -6 through 6, which will cause transposes of down by six semitones through up by six semitones.

2.6.6 *ECHO

*echo is a debugging facility which allows one to display messages on the computer console. Its use should be avoided in production, because it causes extra non-productive work for the computer, and could limit available polyphony. The template for *echo is

*echo (<expression>) “ text “

Here, <expression> is any expression that can be evaluated to a number, and that number will be printed.

“ text “ is any characters which get printed along with the expression.

In one *echo statement, the (<expression>) and/or “ text “ parts may be repeated any number of times to create the desired output. The entire *echo statement is re-evaluated frequently, and if the resultant output is changes, the new result is displayed on the terminal. An example might be

*echo “Swell: “ (swell) “ Great: “ (great)

and the resulting output might be something like

Swell: 17 Great: 2

2.6.7 *SYSTEM

*system can be used to execute a operating system command. The template is

*system (<expression>) “command”

Here, <expression> is any expression that is evaluated. If this expression ever changes from zero to non-zero, the “command” will be sent to the operating system as a command. If this command takes a long time to execute, the sound will likely be interrupted.

As an example, we'll show how to set up a button which will cause the computer to turn off.

First, set up the input to be used. As a precaution against accidental operation, we'll use an obscure controller to trigger the power off. If the sound engine is being fed from a Micro-MIDI system, this can be done by including the following in an appropriate input board definition:

*one_of_n *bit=23 *mch=13 *controller=123 *range=0,79

That's pretty obscure. When input 23 is enabled, it will send a controller number 123 on channel 13 with a value of 79. This is highly unlikely to happen accidentally.

Now, in the sound engine configuration file, place the following:
*system (c13c123 == 79) “shutdown -h now”
If we try this, something like the following is likely to occur: (?)

3 Configuring the Sound Engine

There are any number of ways to configure a Linux system so that the sound engine will run automatically when the system is booted.
(example here)
When the sound engine is run, arguments can be added to the command line to control its behavior. All the possible arguments are covered in this section.

-d Debug information

The -d should be followed by a hexadecimal number which represents the debug information which is desired. This number should be the sum of the following desired items:

1

The sound engine will continuously display the number of notes simultaneously sounding at any one time, the instantaneous polyphony.

2

The letters n, f, and c will be printed for incoming MIDI messages of notes on, notes off, and controller settings respectively.

4

More detailed MIDI information than 2

100

Detailed configuration file parsing information.

An example is “-d1”.

-f Frame size

The -f option may be used to alter the frame size. The default is 44 samples. This is the number of samples which are output in one millisecond, which means that the sound engine calculates one millisecond of data at a time. Making the frame size larger would result in less work for the computer, but would result in increased latency. The default would be “-f44”.

-l Latency

I don't know what this does.

-m Set MIDI device

If no other measures are taken, the MIDI device which the sound engine will use is “hw:0,0”. If this is not the correct MIDI device, use the -m option to specify the correct device. For example, “-mhw:1,0” (leaving off the quotes).

-p Set PCM device

The PCM device is the device which the sound engine uses to output the sound. Unless altered elsewhere, the default PCM device which the sound engine will use is “hw:0,0”. If this is not the correct PCM device, use the -p option to specify the correct device. For example, “-phw:2,0” (leaving off the quotes).

-v Verbose

-vv Very verbose

The -v option will cause the sound engine to display extra information. The -vv option will cause the sound engine to display even more information.

-e Switch endian

If a particular computer and sound card combination produces noise instead of music, the endianness may be backwards. Try adding the -e option.

This might be necessary when a USB sound card is used on a power-pc computer.

-s Switch endian using software

In some cases, the -e option may not correctly switch the endianness of the sound. Using the -s option accomplishes the same thing in a way that consumes more CPU time.

-t Test syntax

Using the -t option allows one to check for the proper syntax of the configuration file, without having correct license file, or the sound files. The sound engine will not play when the -t option is given.

-E Echo

The -E option will cause the sound engine to repeat (echo) any information received on the MIDI in port to the MIDI out port.

Options should be separated with spaces, and precede the name of the configuration file. Thus, a sound engine command with several options might appear

/home/artisan/se -mhw:1,0 -phw:2,0 -d2 -vv /home/artisan/myconfig

THE NEW ARTISAN SOUND ENGINE ADDENDUM:

AUDIO CHANNELS IN THE ARTISAN SOUND ENGINE

The keywords for audio channels are as follows:

SINGLE CHANNEL OUTPUT
*channel 1 = audio channel 1 (numbered on the audio output plug in very small print)
*channel 2 = audio channel 2 (numbered on the audio output plug in very small print)
*channel 3 = audio channel 3 (numbered on the audio output plug in very small print)
*channel 4 = audio channel 4 (numbered on the audio output plug in very small print)
*channel 5 = audio channel 5 (numbered on the audio output plug in very small print)
*channel 6 = audio channel 6 (numbered on the audio output plug in very small print)
*channel 7 = audio channel 7 (numbered on the audio output plug in very small print)
*channel 8 = audio channel 8 (numbered on the audio output plug in very small print)
NOTE in Single Channel mode you cannot play pure stereo ranks. You must use stereo mode for that

STEREO OUTPUT
*stereo 1 = audio channels 1 and 2 in stereo left and right
*stereo 3 = audio channels 3 and 4 in stereo left and right
*stereo 5 = audio channels 5 and 6 in stereo left and right

*stereo 7 = audio channels 7 and 8 in stereo left and right
NOTE in Stereo mode you are defining channel pairs so only the first channel of the pair is designated

DIATONIC STEREO OUTPUT
*diatonic 1 = audio channels 1 and 2 with C and C# sides of the rank alternating left then right
*diatonic 2 = audio channels 1 and 2 with C and C# sides of the rank alternating right then left
*diatonic 3 = audio channels 3 and 4 with C and C# sides of the rank alternating left then right
*diatonic 4 = audio channels 3 and 4 with C and C# sides of the rank alternating right then left
*diatonic 5 = audio channels 5 and 6 with C and C# sides of the rank alternating left then right
*diatonic 6 = audio channels 5 and 6 with C and C# sides of the rank alternating right then left
*diatonic 7 = audio channels 7 and 8 with C and C# sides of the rank alternating left then right
*diatonic 8 = audio channels 7 and 8 with C and C# sides of the rank alternating right then left
NOTE in Diatonic mode you are defining channel pairs but you may define either channel as the starting channel and the other channel will be assumed to follow. This new feature allows you to do things such as designate a string and it's celeste to play from opposing speakers making for a far better ensemble.

Be certain to adjust your *shades statement to reflect any changes you make to your audio channels.

The shades statements remains the same except that you must remember to identify both channels of a stereo or diatonic set together so that they are not broken up into two separate expressions. Any channel not appearing in an expression statement will remain unexpressed.

Here is an example of a shade statement that is to effect control over channels 1, 2, 3, and 4.

*shades 1 2 3 4 ((c7c7 != -1) * (c7c7/2 - 50))

Now when Midi channel 7, Midi controller 7 is sent then Audio Channels 1,2,3,4 will respond to the change.

This statement will place channels 1,2,3,4,5 in the main and channels 6,7,8 in the solo:

*shades 1 2 3 4 5 ((c7c7 != -1) * (c7c7/2 - 50))

*shades 6 7 8 ((c8c7 != -1) * (c8c7/2 - 50))

We're of course assuming that you are sending the volume control for Main over Midi channel 7 and the volume control for Solo over Midi channel 8, both sending Midi controller 7, the accepted General Midi controller for volume.

QUICKSTART ADJUSTING VOLUME OF RANKS:

Adjusting the volume of an individual rank is done by raising or lowering the numeric value of the *level statement on a given rank.

Example:

*rank rprincipal 8 “/ranks/principal8.arnk”
*stereo 1
*level 2.5
*trem 1

NOTE: If no *level statement is present for a rank it means that this rank will play at the originally recorded volume. You may add a *level statement to any rank in order to make changes to that original volume.

When adding the numeric value to the *level statement bear in mind that less is a good thing. In most cases you will want to lower the volume rather than raise it. If you try to raise it past the point where digital clipping would occur, the sound engine software will automatically limit the outcome to keep this from happening. So, in practice, always remember that in order to make a given rank louder it is more advisable to lower the volume of other ranks in relation.

For the value statement you may use decimals as in the example above. Never raise the volume of any rank or trap more than 5. You may lower the volume of any given rank as much as you like.

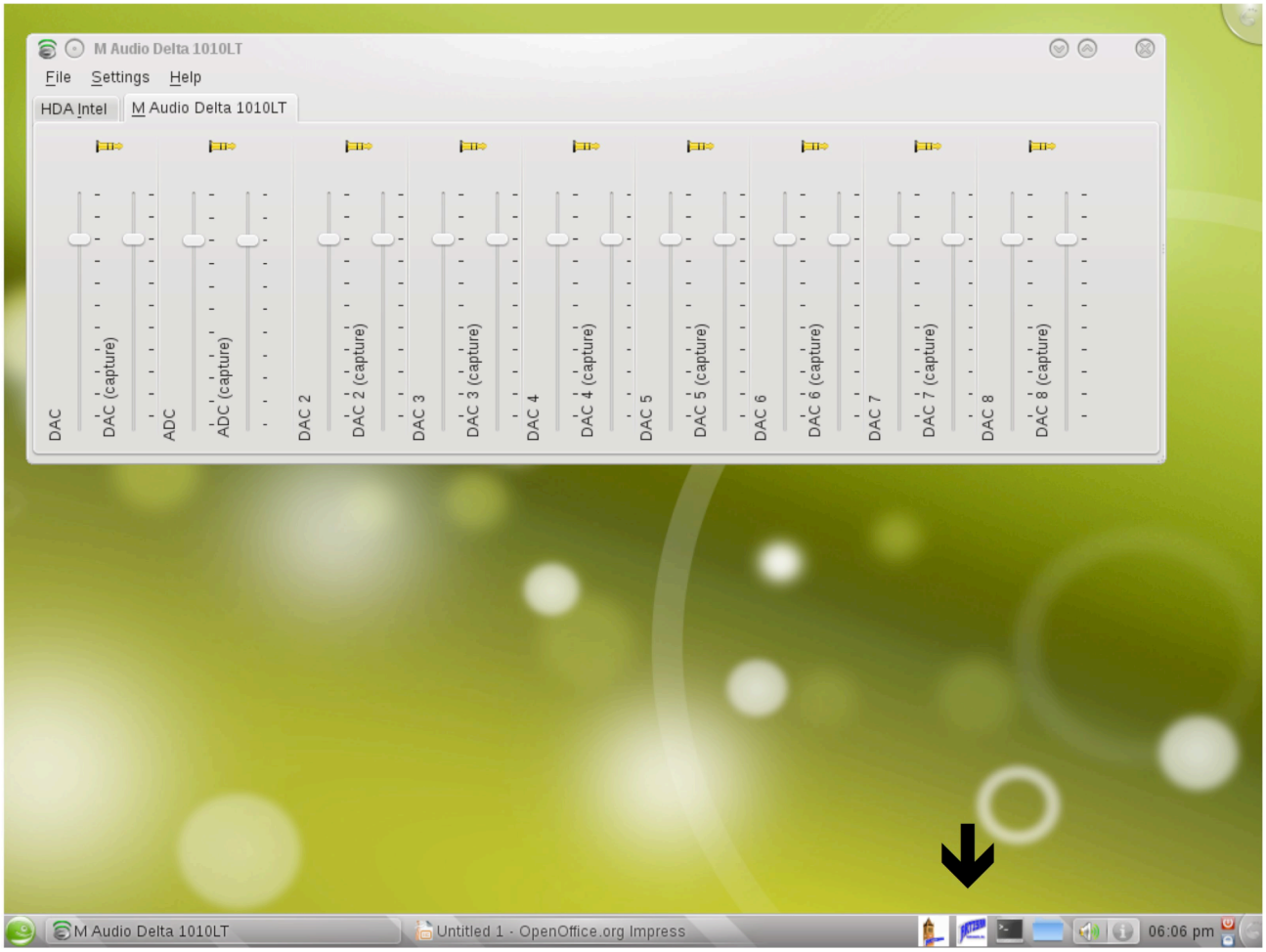
To adjust the volume of a given audio channel use the audio sliders found in the graphical interface on your desktop. Bear in mind that if you have stereo ranks, it will require that you raise or lower two sliders according to the particular stereo channel set you have defined.

ARTISAN REGULATION PROGRAM INSTRUCTIONS

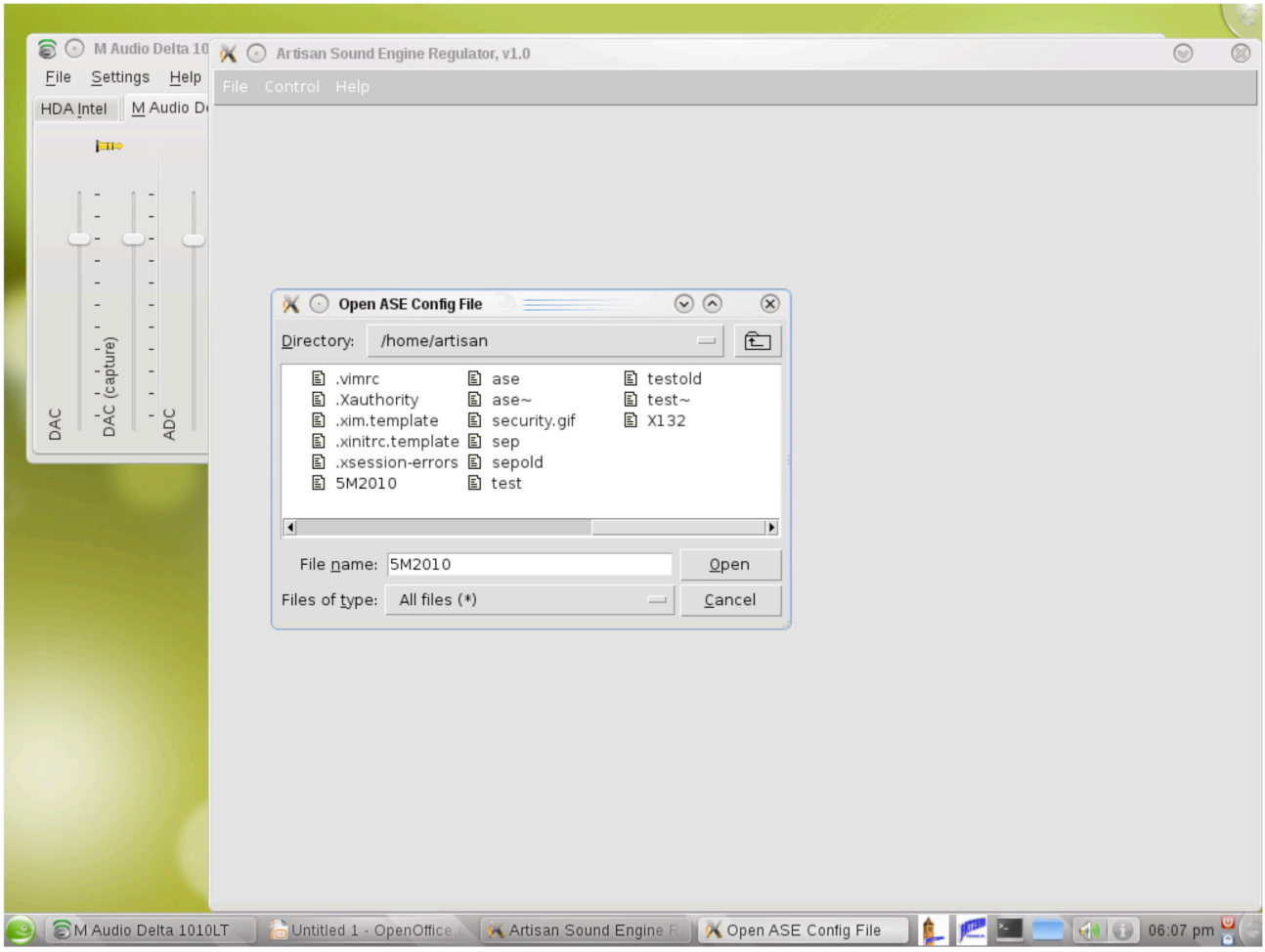
On the newest sound engines from Artisan we include a graphical program for the regulation of rank volume, tremulants, audio channels, etc.

To use this program you must have a monitor, mouse, and keyboard attached to your sound engine.

To start the program look for the small Artisan icon on the desktop as shown below. Single click that icon.



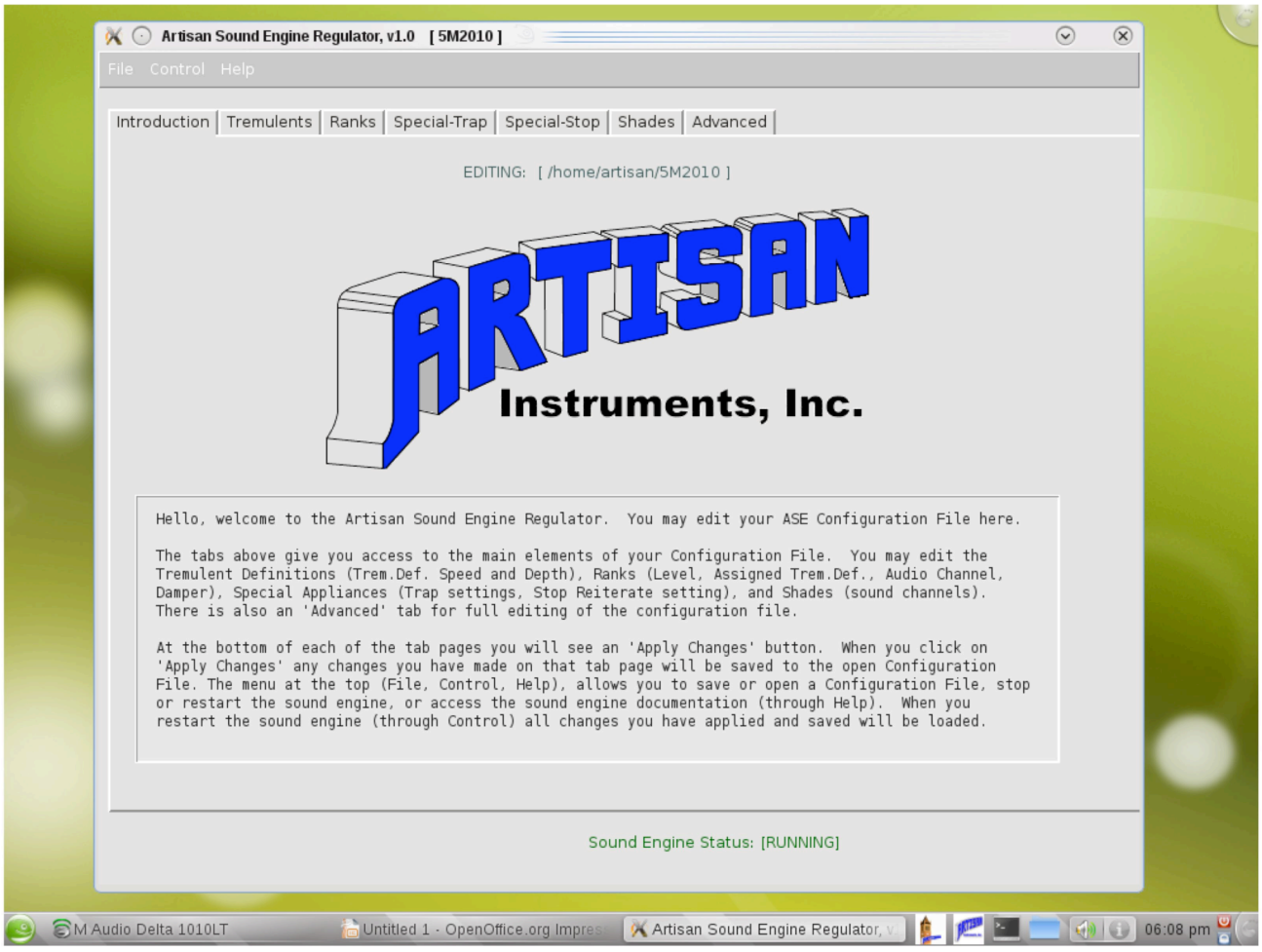
This will open the start screen which will allow you to select the configuration file for your sound engine. Artisan will supply that filename for your instrument.



Find your file in the list and click on the file, then click “OPEN”

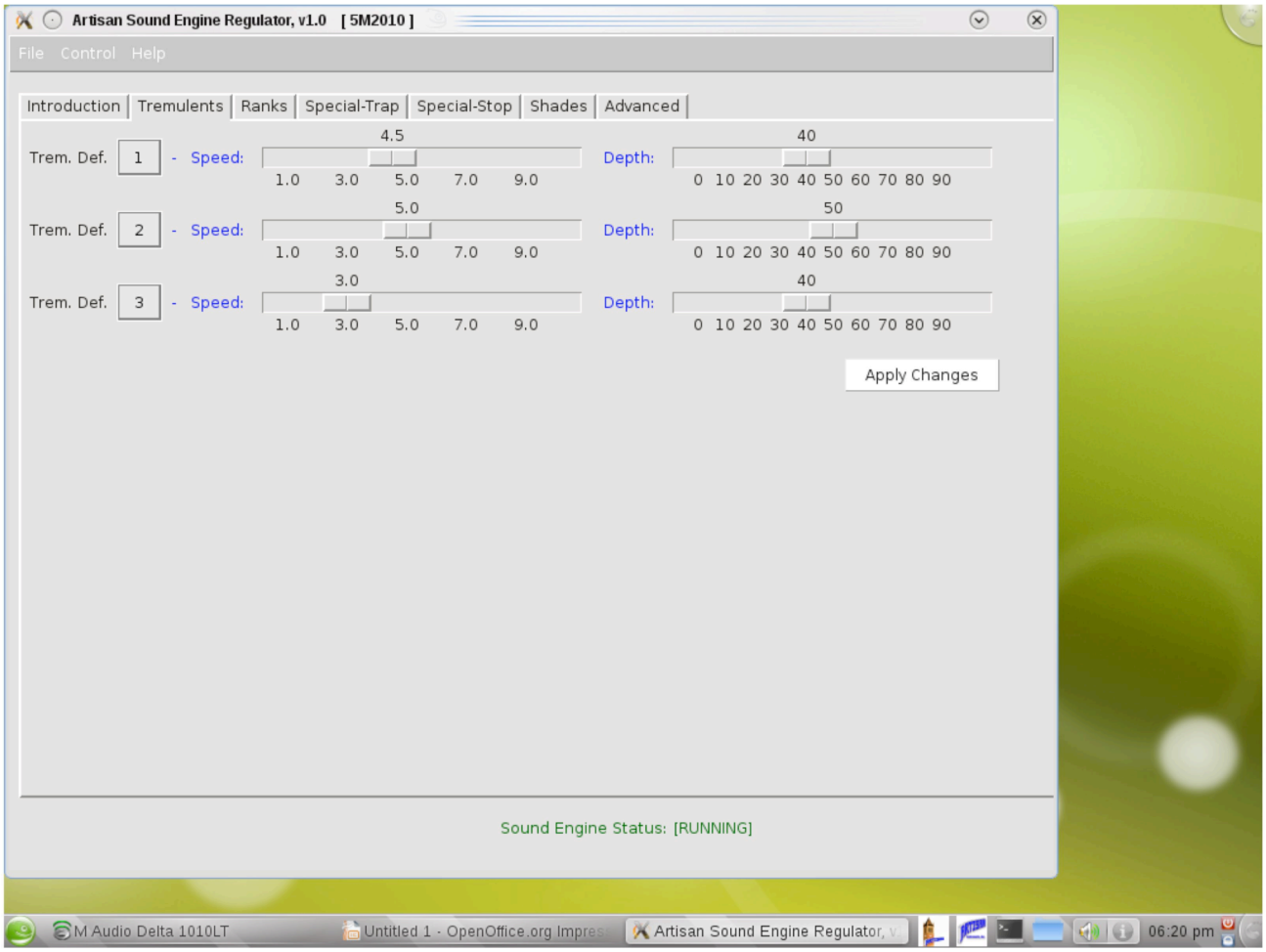
You will then be presented with the regulation program main screen. Read the instructions listed and look for the tab links as follows:

TREMULANTS – RANKS – SPECIAL TRAP – SPECIAL STOP – SHADES - ADVANCED



These tabs will allow you to make the appropriate changes to your organ. At the bottom of the screen there will be a prompt to let you know whether or not the sound engine program is currently running. If it is running you may click on “CONTROL”
Then you may stop or start the sound engine from that pull down menu.

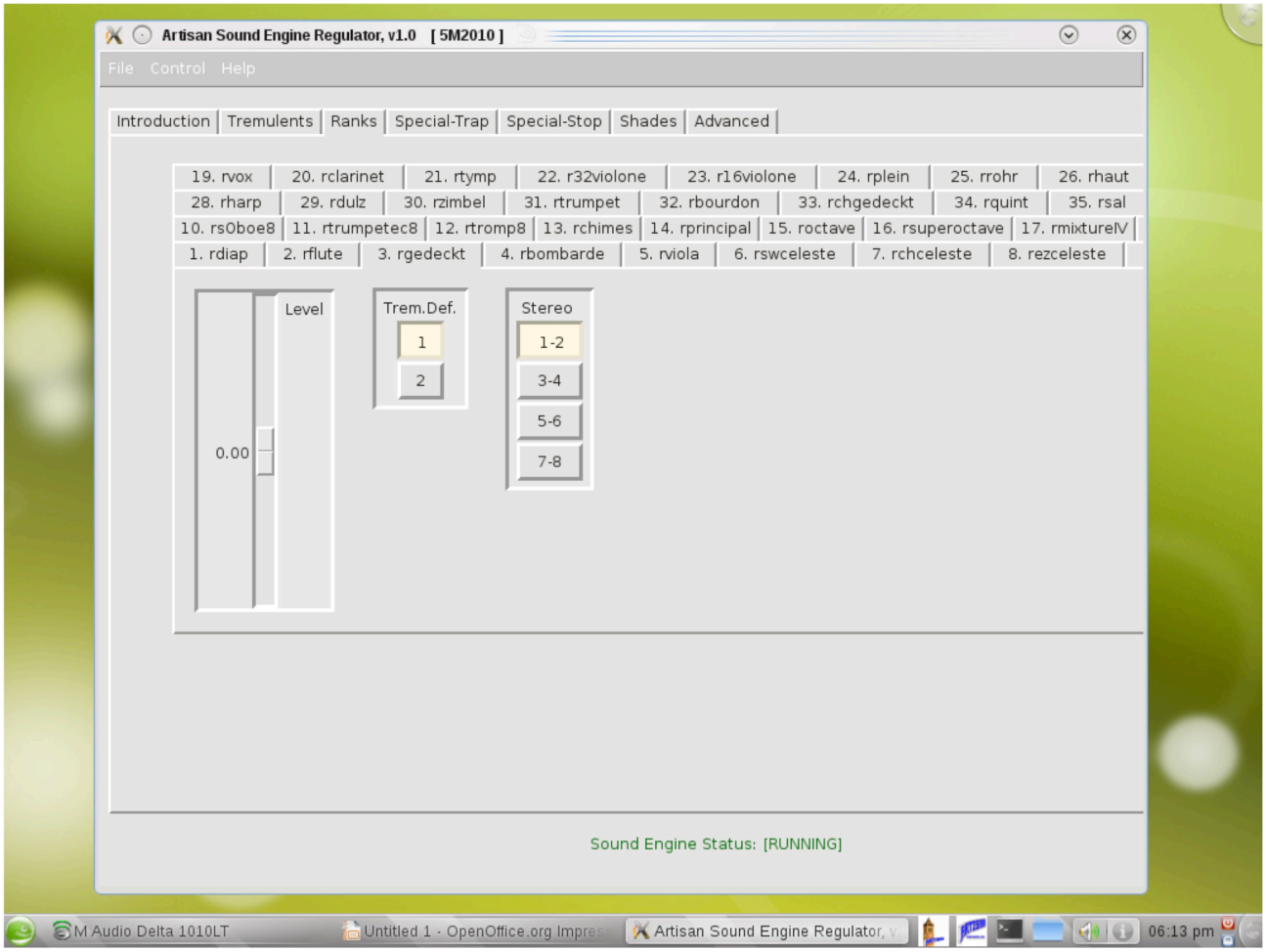
Select Tremulants first to see a list of available tremulants for your organ.



You may adjust the sliders on each tremulant to increase or decrease the speed and depth of each tremulant.

If you make changes don't forget to clcik on the "Apply Changes" button. If you do forget you will have another opportunity before quitting the program.

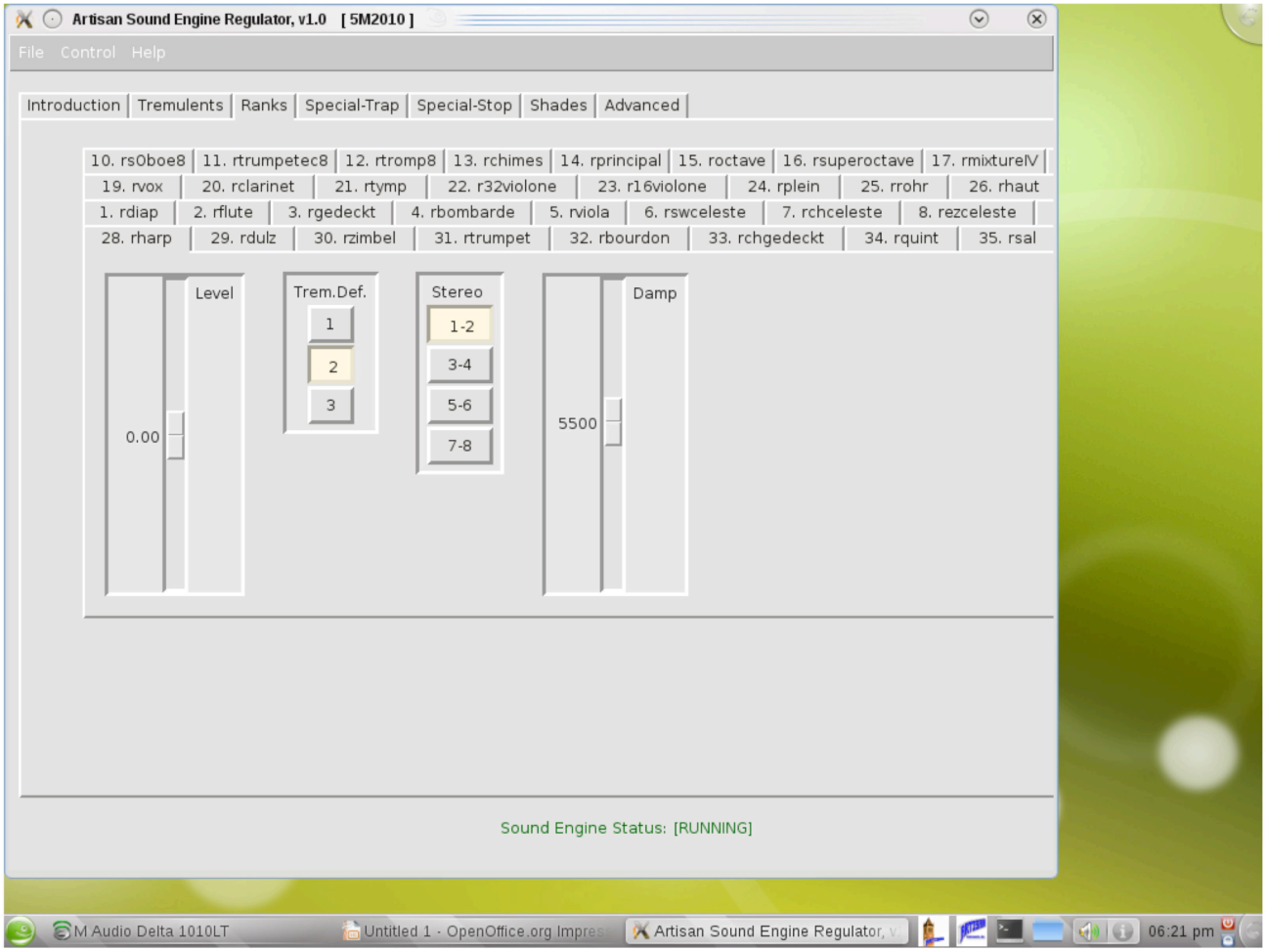
Next tab is the RANK screen.



Here you will be able to adjust the volume of the rank and each note within the rank. To access the volume of each note you must double click the rank name.

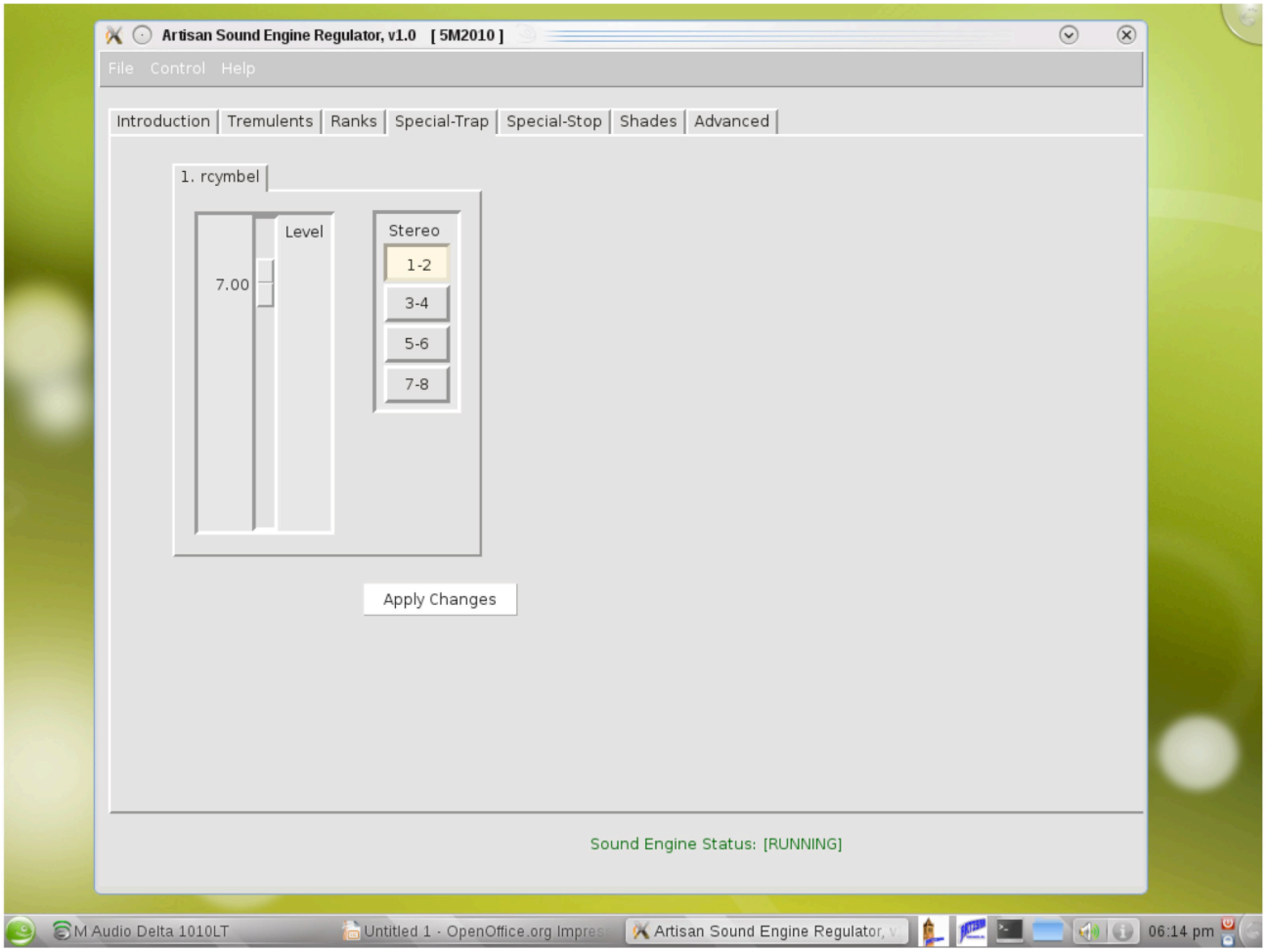
You can also assign the rank to a tremulant if desired and then select the audio channel through which this rank will be played.

If the rank contains a special parameter such as damper length (in things like piano, harp, etc.) you will also be presented with a slider for adjusting those parameters.



Feel free to experiment, you can always change things back if you do not like the results.

The next screen is the Special Trap screen. It will present you with any trap stops or struck non tuned percussions stops in your organ. These include things such as Cymbals, Drums, etc.

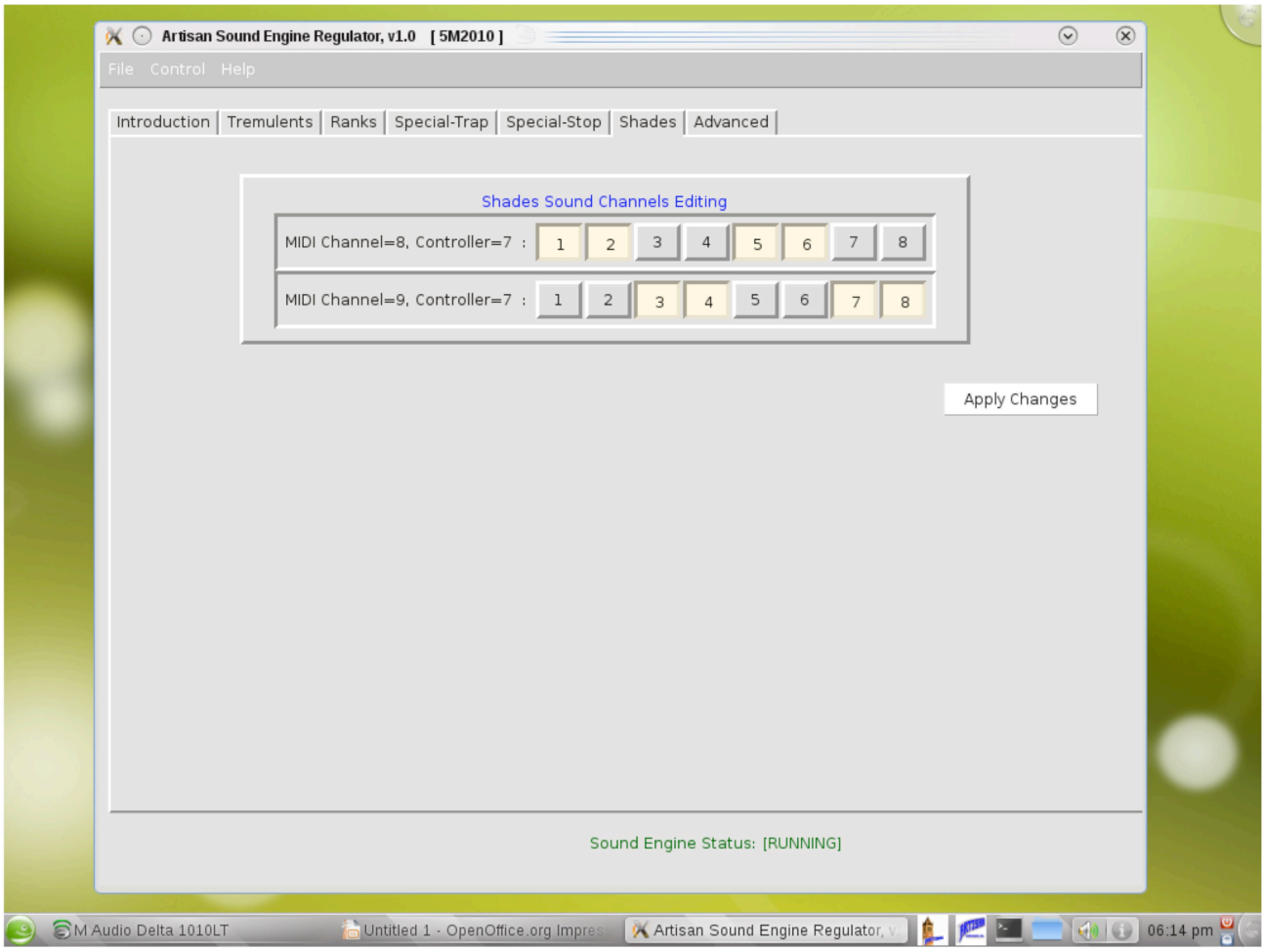


You can select the volume level and audio channel of each of these stops.

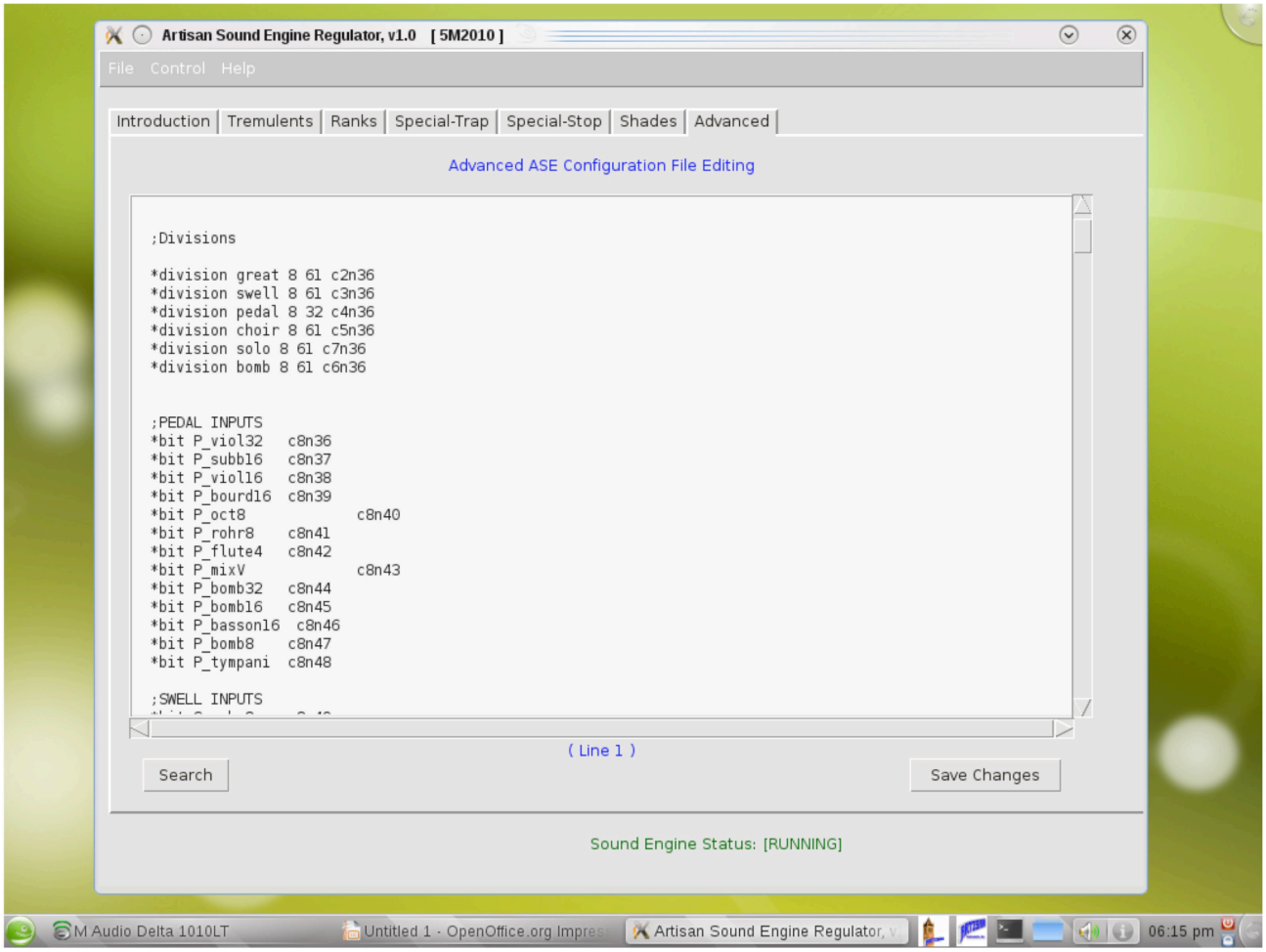
The SPECIAL STOP screen will allow you to regulate things like the rate of reiteration on a Xylophone, Marimba, Orchestra Bell, etc. if your organ contains these stops.

Finally we reach the SHADES tab.

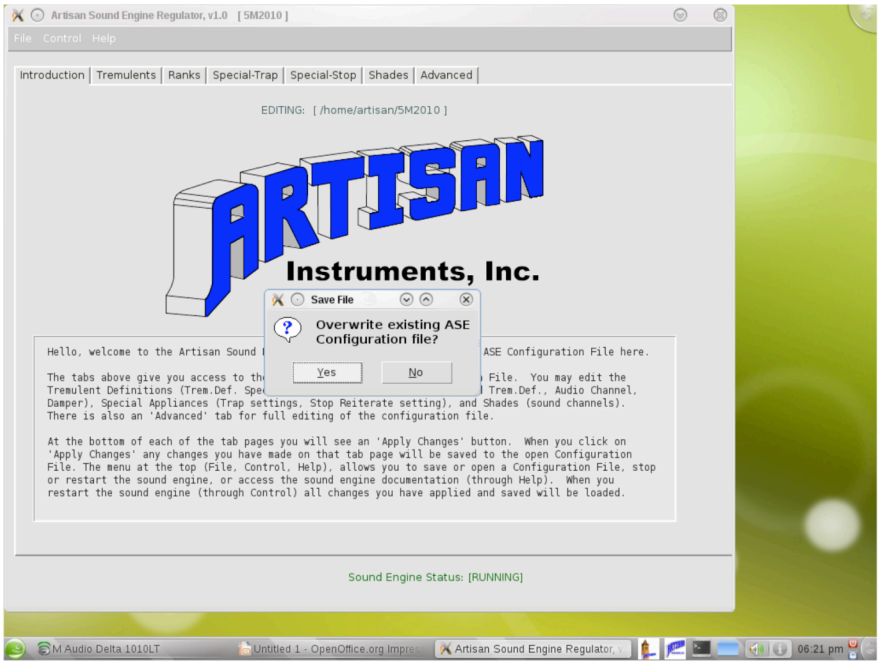
This is where you assign your expression pedals to the appropriate audio channels according to the way you have set your audio output in the RANK tab.

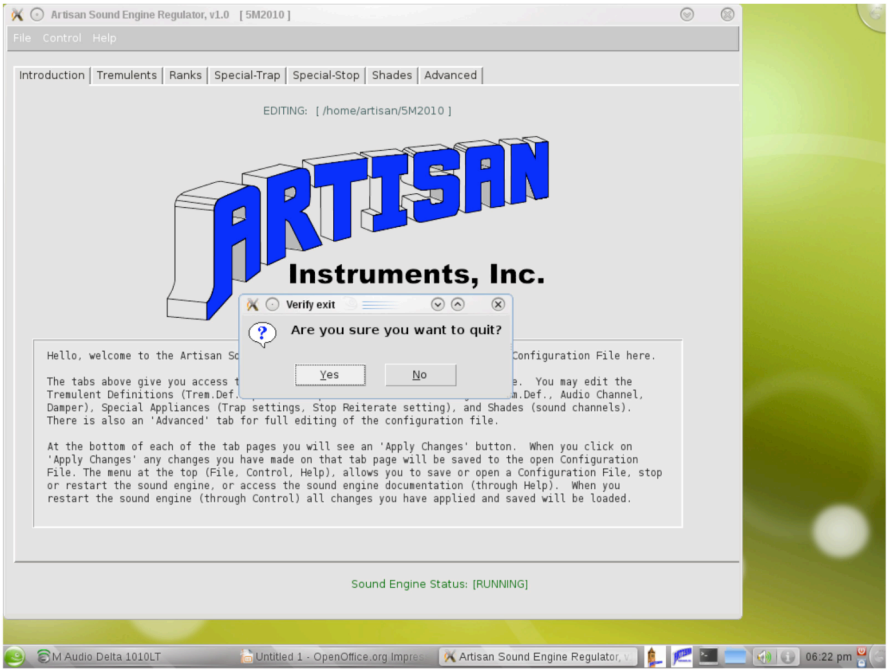


After all of your regulation work is finished you can see the actual changes made to your configuration file under the ADVANCED tab.



You may now restart the sound engine by clicking on the CONTROL menu.
Finally after all changes are made you can save your file and exit the program.





For any assistance please call Artisan support (425) 485-6743