CUSTOM SCALE EDITOR

Hπ INSTRUMENTS

Aaron Andrew Hunt

1. Installation

Mac OSX

CSE does not require a MIDI interface; however, to upload tunings to TBX1 you must have a MIDI interface and have properly configured your equipment in Audio MIDI Setup, which is in the following location:

Your Hard Drive > Applications > Utilities > Audio MIDI Setup

It's a good idea to drag the Audio MIDI Setup icon to your dock so that you can easily access this application, as you will need to do so periodically when working with any audio software.

The Mac version of CSE will load Soundfont .sf2 files located in either of the following locations:

Your Hard Drive > Users > Your User Name > Library > Audio > Sounds > Banks
Your Hard Drive > Library > Audio > Sounds > Banks

Windows

To have proper display of note names in CSE, the ARIELL font must be in the Fonts folder. CSE will attempt an installation at startup, but this may cause an error. If installation is unsuccessful, simply right-click the ARIELL.ttf file and select *Install Font*, or drop the file into the Fonts folder, which is in:

My Computer > Control Panel > Fonts

CSE does not require a MIDI interface; however, to upload tunings to TBX1 you must have a MIDI interface and have properly configured your equipment. Visit your MIDI setup here:

My Computer > Control Panel > Sounds and Audio Devices > Audio

2. Basics

Built-in Help

CSE has built-in help windows for many of its features. Most, but not all, of the information given here is duplicated there. Click the question mark icons. Menu Help > Documentation PDF opens this document.

Bug Reporting and Feature Requests

Should you experience a crash, a copy of the file is saved for recovery and a bug reporting dialog appears. You can also use the menu item > Report Bug at any time. A list of reported bugs and feature requests is maintained at hpi.zentral.zone/reports/cse

Setting Pitch Bend Range

TBX1 sets the pitch bend range of devices connected to its MIDI OUT to +/- 1 semitone; however, some devices may require the default MIDI pitch bend range of 2 semitones. Other cases such as MIDI guitar synthesizers require pitch bend ranges of +/- 24 semitones. Use the menu TuningBox > Set Bend Range before programming tables for bend ranges other than +/- 1 semitone.

Programming a Tuning

Tunings created with CSE must be uploaded to TBX1 to be used. Follow these steps:

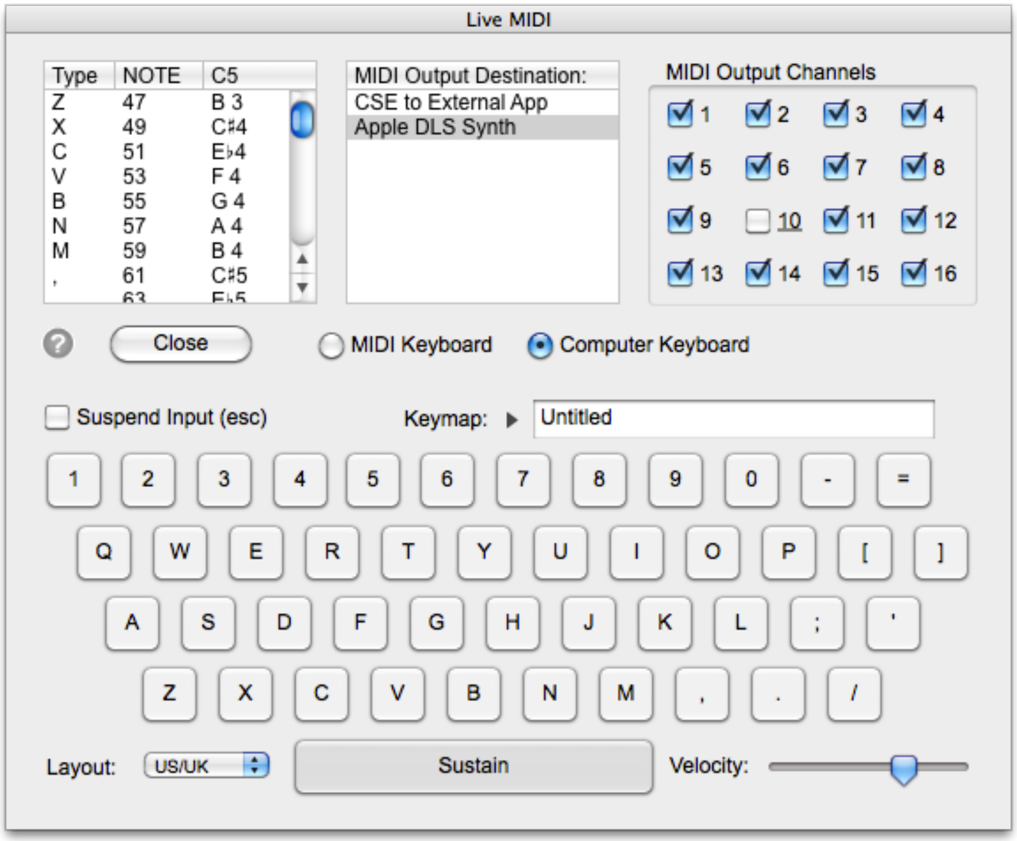
1. connect your MIDI interface to your computer, have it properly installed, etc.
2. connect MIDI OUT from the interface to MIDI IN on TBX1
3. boot CSE and load your tuning
4. choose menu item MIDI or type Command M and select the interface as the output port
5. choose menu item TuningBox and choose Program, or type Command P
6. select the options you want and program the tuning. TBX1 will display PROGRAMMING
7. unplug TBX1 from the interface and plug it back into your setup

NOTE: if MIDI devices are connected or disconnected while CSE is running, you must choose MIDI > MIDI Routing and click the Rescan button.

File Importing and Exporting

CSE can import and export a number of full range tuning file formats, including Scala .scl, VAZ / Anamark .tun and .msf, Native Instruments Kontakt .txt and Absynth .gly, and Max Magic Microtuner .mtx 12-tone tuning file formats are also supported, including Cubase / Nuendo .xml, and Native Instruments Pro53 .p5m. New formats can be added upon request. Let me know what you want, and I will do my best to add it to the software.

Live MIDI Input (CSE Pro)



Type	NOTE	C5
Z	47	B 3
X	49	C♯4
C	51	E♭4
V	53	F 4
B	55	G 4
N	57	A 4
M	59	B 4
,	61	C♯5

MIDI Output Channels: [✓ 1] [✓ 2] [✓ 3] [✓ 4] [✓ 5] [✓ 6] [✓ 7] [✓ 8] [✓ 9] 10 [✓ 11] [✓ 12] [✓ 13] [✓ 14] [✓ 15] [✓ 16]

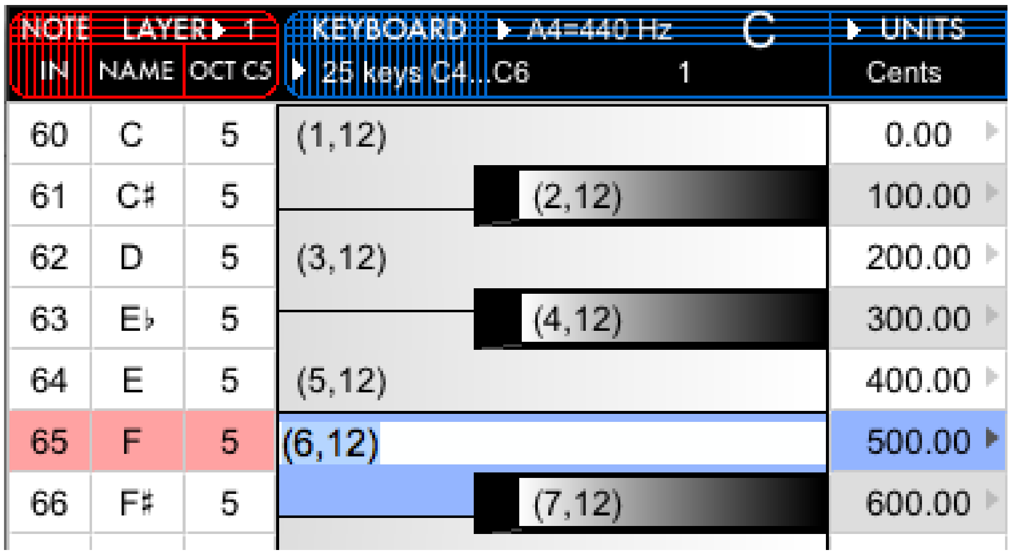
MIDI Keyboard [✓ Computer Keyboard]

The menu item MIDI > Live Input opens a window which allows you to use CSE as a virtual Tuning Box, routing incoming and outgoing MIDI to and from external applications without the need for TBX1 to be connected into the MIDI interface, or using the computer keyboard as a virtual MIDI keyboard. Mac Users will see two virtual ports in the MIDI connections lists. Windows users must install the free utility MIDI Yoke in order to see virtual MIDI connections.
http://www.midiox.com/myoke.htm

NOTE: Only registered CSE Pro license holders can unlock this feature using an internet connection and a valid name and license code.

3. Interface Basics

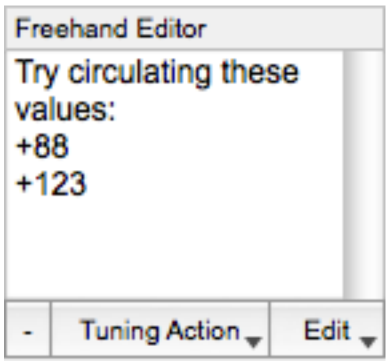
Keys List



NOTE	LAYER ▶ 1		KEYBOARD ▶ A4=440 Hz C	▶ UNITS
IN	NAME	OCT C5	▶ 25 keys C4...C6 1	Cents
60	C	5	(1,12)	0.00
61	C♯	5	(2,12)	100.00
62	D	5	(3,12)	200.00
63	E♭	5	(4,12)	300.00
64	E	5	(5,12)	400.00
65	F	5	(6,12)	500.00
66	F♯	5	(7,12)	600.00

All tuning entries for every key are listed in a scrolling list. Note that the list is numbered from top to bottom, and the pitches thus rise from top to bottom. If this seems backwards, consider that listing from top to bottom is the standard way to list anything which is ordered, and Western musical scales have been listed this way since the first recorded scales of ancient Greece.

Freehand Editor



Freehand Editor

Try circulating these values:
+88
+123

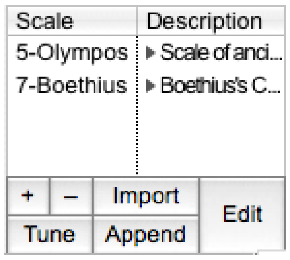
The freehand field is a text edit field which allows you to type any text you want to cut and paste. It is most useful for cutting and pasting text from other programs into CSE to create scales and tuning entries. Use the **Tuning Action** button to create a scale from the field contents, and use the **Edit** button to convert the field contents to various entry types. For example, you may paste a list of values from another application which you want to tune in CSE as Hz values. The **Edit** button allows you to do this by choosing **Interpret as Hz Entries**. A list of possible field formatting is below.

Interpret as Hz Entries

- **Interpret as Unit Entries**
- **Replace Tabs with Returns**
- **Replace Commas With Dots, Replace Dots With Commas, Swap Dots and Commas**

Chapter 5 gives details on tuning entries.

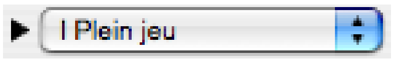
Scale List



Scale	Description
5-Olympos	▸ Scale of anci...
7-Boethius	▸ Boethius's C...

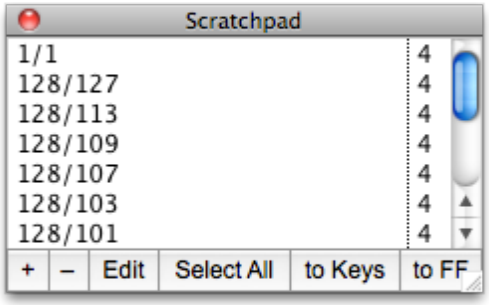
Below the Freehand field is a list of imported scales. You may want to use more than one Scala file to create your tuning. Click Import to add a .scl file to the list, or create a new scale from scratch, or transfer the contents of the Freehand field using the Scale button. Once scales are in the list, they can be tuned to the keyboard by selecting an area and choosing Action > Tune Scale.

Instrument List



Select a playback instrument from the instrument list. On PC, this is a GM sound set, but output can be sent to any external port if MidiYoke is installed. On Mac, this list contains Quicktime Instruments plus any soundfonts located in ~ Library > Audio > Sounds > Banks. The soundfont must have the extension .sf2. On Mac, there is a triangle next to the popup menu as shown above, which when clicked opens a window giving control of Mac OSX Synthesizer parameters, including the sample set, patch, default duration and velocity for automatic notes, and AU parameters Cents Offset, Reverb, and Volume.

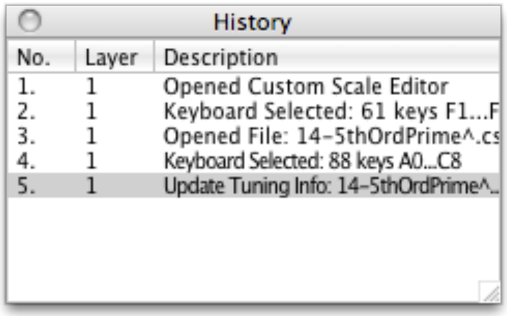
Scratchpad Window



```
1/1      4
128/127  4
128/113  4
128/109  4
128/107  4
128/103  4
128/101  4
```

When the cursor is not in the Freehand field, copying and pasting is done using the Scratchpad window, which can only contain valid tuning entries. Think of it as a dedicated tuning clipboard. Clicking the **to Keys** button pastes selected entries in the list to the keyboard. You can also transfer contents to the Freehand editor with the **to FF** button.

History Window



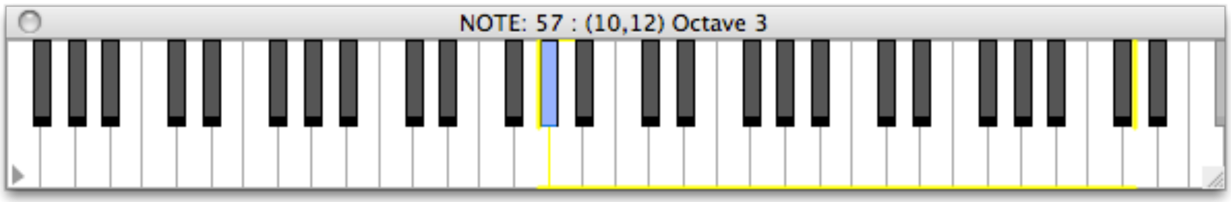
No.	Layer	Description
1.	1	Opened Custom Scale Editor
2.	1	Keyboard Selected: 61 keys F1...F
3.	1	Opened File: 14-5thOrdPrime^.cs
4.	1	Keyboard Selected: 88 keys A0...C8
5.	1	Update Tuning Info: 14-5thOrdPrime^...

Many user actions are stored in a history list, which is viewable in the History Window. Actions can be undone with Command-Z or previous states can be accessed by clicking on the history list. Note that most, but not all actions are stored in the history list. If you find an action not stored which you want stored, you can always file a feature request.

Mouse and Computer Keyboard

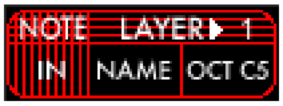
Click a row to select a key. Shift Click selects a contiguous range of keys. Mac Option (PC Alt) Click selects a discontiguous range. Control Click selects a key in every octave. Up and down arrows navigate the keys when a single key has been selected. Spacebar toggles Sustain Mode ON / OFF, useful for testing out chords.

Floating Mouse-Click Keyboard (CSE Pro)



Choose menu item View > Floating Keyboard to open a window allowing you to navigate the selected keyboard range by clicking on keys to hear pitches. Use the triangle at the lower left to toggle between the entire MIDI range and your selected keyboard size.

4. About Layers



A layer is a tuning of 128 notes. There are 4 layers per table, resulting in 512 notes per table, Layers correspond to input channels on TBX1. When a table having multiple layers has been uploaded to TBX1, each layer can be accessed from the TBX1 by a MIDI controller transmitting on the channel with the same number as the layer number, mod 4. Layers and MIDI input channels correspond as follows:

Layer 1 = MIDI input channels 1, 5, 9, and 13
Layer 2 = MIDI input channels 2, 6, 10, and 14
Layer 3 = MIDI input channels 3, 7, 11, and 15
Layer 4 = MIDI input channels 4, 8, 12, and 16

The Keyboard Size and the Key (as indicated in the header) are linked to the current Layer (1-4). These are selectable by popup menus.

NOTE: Layers have nothing to do with TBX1 output channel switches

5. Tuning Entries

A tuning entry is an expression which represents a tone in a number of different forms as follows.

Basic Entries

Decimal values such as 1.2365256341287

Interval Ratios a:b, where a < b, such as (2:3)

Tone Ratios a/b, where a > b, such as (3/2)

Degrees of an ET, or equal division of the octave, in set notation, where the first scale degree is numbered 1, such as the second scale degree of twelve tones per octave (2,12)

Exponents, such as (3^7/2^4) or the ratio of 3 to the 7th power to 2 to the 4th power.

+ and - Units values (default unit is Cents).

Frequency values in Hz, such as f=261.6255653, which may also be entered in the form f=<CodeSnippet> (see below). The available frequency range is 7.9430 Hz to 12911.4169 Hz.

Result Range

Tuning entries automatically place the resulting value of the entry (x) within the range 1 <= x < 2, that is, within an octave. If you want to return a value which is not automatically placed within this range, you can add the character @ to the start of the entry. This spiral-like character tells the parser that the tone is allowed to spiral past the octave threshold. For example, the entry (3/1) returns 1.5, the justified value of (3/1) = (3/2), while the entry @(3/1) returns 3. The example using exponents given above (3^7/2^4) could just as well be written (3^7/2).

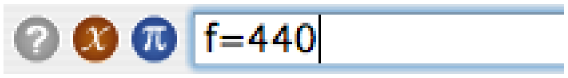
Operators

+ and - are used for units values, such as +35.2 or -4.3, used alone or placed AFTER any other expression. For example, (1,12)+3 adds 3 units to the first degree of 12ET. (7:8)-5.3 subtracts 5.3 units from the ratio 7:8. The tuning value depends on currently selected Tuning Unit. The default unit is Cents (1200 Units, 2/1 Ruler).

> ***NOTE:*** *+ and - cents must always be at the END (right side) of the expression. You will see that the transpose function does this automatically.*

Multiplication and division mean transposition up or down respectively, for example (1.232)*(2:3) is the decimal fraction 1.232 transposed up by the interval 2:3. for example (4/3)/(1.112) is the tone 4/3 transposed down by the decimal fraction 1.232. Note that the / between the segments means 'transposed down by' while the / in the 4/3 segment does not indicate transposition. Parentheses must be used when there are 2 terms and some operator, as shown above. The command is parsed according to the segments which are separated by an operator such as * or /. Parentheses define segments and cannot be nested, except inside code snippets (see below).

Entry Fields


f=440

There are several places to type in tuning entries. The first is the Main entry field, which has three small circles directly at its left: (?) (x) and (pi), which stand for Help, Functions, and Constants, respectively. Use the (x) and (pi) circles to add known functions and constants to the main entry field, and then replace the variables with the values you want to use. The Main entry field restricts keystrokes only to allowable keys in allowable situations. It may be thought of as the safest way to make a tuning entry. Using this field it is also possible to tune a range of keys to the same value.

Another way to enter tunings is to type into Keyboard fields directly over the keyboard keys in the list. These fields are quick to use and are also safe entry fields; however, they lack the advantage of the Function and Constant menu selections.

Cutting and pasting using either the Main or Keyboard fields uses a special clipboard called the Scratchpad, which appears as a floating window. Tuning entries can be copied and pasted from this window without affecting the contents of the computer's clipboard, which may hold text from anywhere, including other applications. The Scratchpad is a kind of safe clipboard which only allows valid tuning entries to be listed.

In contrast to these safe methods of entry, another possibility is the Freehand field. When the cursor is active in this field, the menu choices change to standard text editing options only. The purpose of this field is to provide the familiar feel of a standard text-editor which is free from the safeguards of the other entry fields. Multiple entries may be input in a quick list using return or enter, and text from this field can be copied, cut and pasted to and from other applications. The text can be turned into a Scale and added to the Scale List, and can be copied to the Scratchpad. Note that characters typed in this field can be anything, not just valid tuning entries. So, it can also be used just to jot down random thoughts.

About Octave Numbers

H-Pi software uses the C based octave numbering system, where middle C is called C4. In terms of tuning, the octave boundaries exist not at C+0 cents to C+1199 cents, but at C-50 cents to C+1149 cents. This results from sample mapping logic which is centered on 12ET pitches, bending samples up or down within the narrow range of a quartertone in order for the sound to remain as close as possible to the original sample. Because of the quartertone offset, tuning entries for pitches around the boundaries may result in octave assignments an octave higher or lower than desired. To get the octave you want, just change the octave number using Command-' or Command-/. You can also precede entries with the @ character to allow them to cycle past the octave limit without changing the octave number.

Constants Basics

Constants are decimal values like {pi} and {e} or {yourconstant} which can be plugged in anywhere in a tuning entry. see the section on Constants for more information.

Functions Basics

Functions are user programmable variable calculators like YOURFUNCTION[x] which can be part of a tuning entry as any other segment. See the section on Functions for more information.

Code Snippets Basics

A code snippet is an entry segment which is a line of BASIC computer programming code enclosed in angle brackets, as in <Your Code>. The code is processed as the statement z = <Your Code>, where z can be either a decimal (x) or a string (s), and the value of the segment returned by your code is z. You can think of this like a fancy calculator with all the expected math functions as well as string functions and built-in tuning functions. A standard language reference for code snippets is found in the Appendix.

Of the many uses for code snippets, some basic uses are:

1. Perform calculations with tuning functions and nested parentheses directly in a single statement, such as: `<CentsToDecimal(round(1200*log2(46573))-34.6)>`
2. Dynamically link a key to an imported scale or scales. For example, you have imported a scale (scala) file called FirstScale. You can link a key to, say, the third tone in this scale, with the code snippet `<scX("FirstScale", 3)>` Then if you want to edit your scale and change the third tone, the key updates to the new tone without having to copy and paste in a new value.
3. Combine calculations with scales in something like: `<round(310*log2(scX("FirstScale", 3)))>`, or even `<round(310*log2(scX("FirstScale", 3)/scX("SecondScale", 4)))>`, etc.

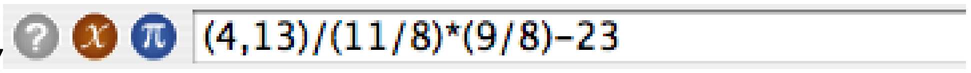
Parsing

Tuning entries are parsed as follows:

1. Constants {c} are replaced with actual values
2. Functions THISFUNCTION[x] are processed and replaced with actual values.

3. Code Snippets are processed and each <CodeSnippet> is replaced with an actual value
4. The remaining basic entry segments are processed and replaced with actual values.
5. All of the values are calculated according to the operators used between the segments, and the final value is returned.

Example Entry


(4,13)/(11/8)*(9/8)-23

This is the 4th degree of 13ET transposed down by 11/8, then transposed up by 9/8, and then transposed down by 23 cents. Up to 32 operations can be strung together for each key.

> ***NOTE***: *Parentheses must be used when there are 2 terms and some operator, as shown above. The command is parsed according to the segments.*

Decimalizing, Rationalizing and Simplifying Entries

Entries can be changed directly into decimal values by using the **decimal** button or menu item, and can be made into ratios by using the **ratio** button or menu item. Entries with multiple ratios can be simplified using the menu **Action > Simplify Ratios**, or using **Command =**. All ratios, which may be anywhere and a mixture of a:b and a/b types, are multiplied or divided together and then reduced. The example tuning entry given above, (4,13)/(11/8)*(9/8)-23.4, simplifies to: (4,13)/(99/64)-23.4, this entry decimalizes to 1.49683231544365002996244129462866113, and rationalizes to 235/157.

6. Global Parameters

The actual pitch resulting from a tuning entry depends on four parameters in addition to the entry itself. These are:

· the Hz frequency value of the note called A4

· the interval from 1/1 up to A

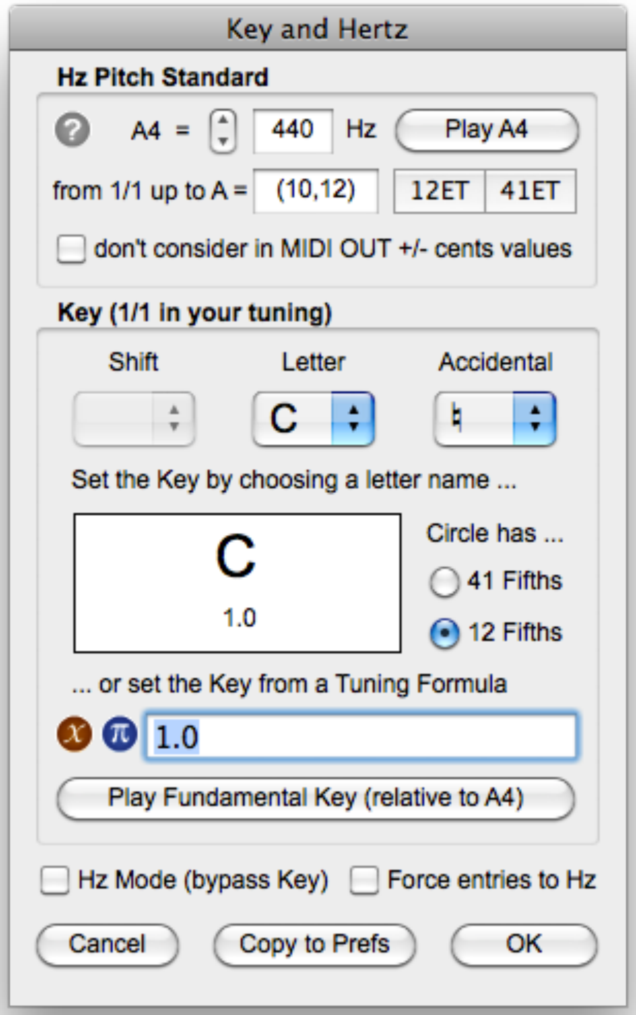
· the number of fifths on the circle of fifths (12 or 41 fifths)

· the global Key

A4 Hz and Interval from 1/1 up to A

Two pitch defaults in Western musical history are C, as a ‘Major Key’ and A, as a ‘Tuning Pitch’. H-Pi software uses these defaults. The default Key is C, (meaning 1/1 is called C), and the default frequency for A is 440Hz. To correlate these two values, the interval between A and C must be known. In CSE, this is defined as the interval *from 1/1 up to A*, defaulting to a Major Sixth in 12ET, or (10,12) = 1.6817928305 ...

Since these default parameters correspond to MIDI defaults, if you don't change them, then the MIDI cents offsets will be +/- 0 ¢ when you tune a scale in 12ET. If you change the parameters and you don't want to see the MIDI cents offsets resulting from your changes, you can check the box *don't consider in MIDI OUT +/- cents values* in the Key and Hertz window. The default values are shown below:


Key and Hertz

Hz Pitch Standard

A4 = 440 Hz

from 1/1 up to A = (10,12)

☐ don't consider in MIDI OUT +/- cents values

Key (1/1 in your tuning)

Shift: (none)

Letter: C

Accidental: ♮

Set the Key by choosing a letter name ...

C

1.0

Circle has ...

○ 41 Fifths

● 12 Fifths

... or set the Key from a Tuning Formula

1.0

☐ Hz Mode (bypass Key)

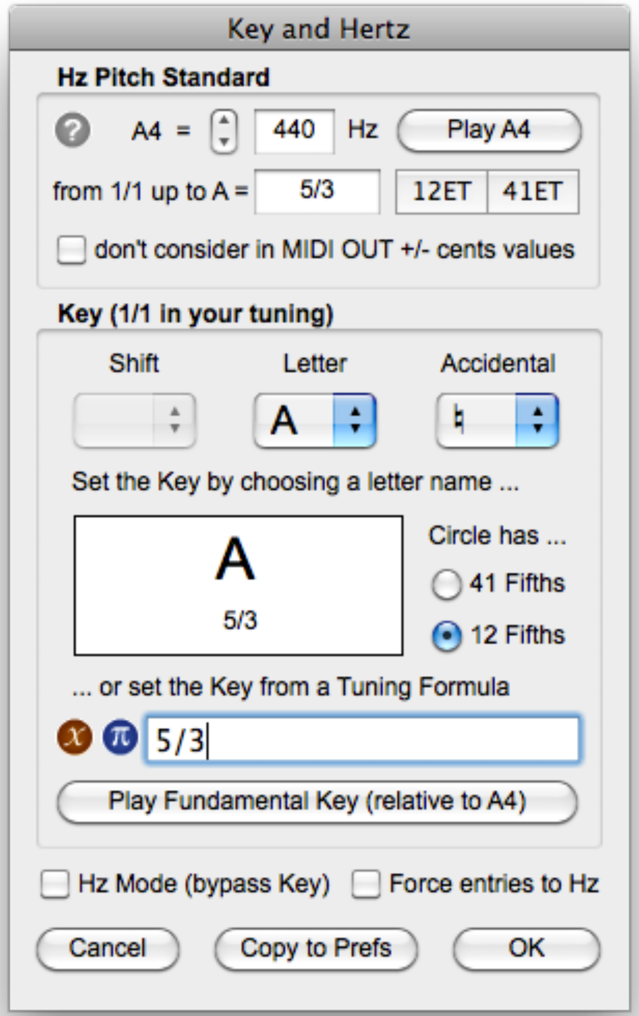
☐ Force entries to Hz

NOTE: the Hertz value of A also depends on the global pitch settings of the synthesizer. For the Hertz setting of .cse files to work correctly, the synthesizer must have its A4 set to 440 Hz.

Key and Fifths on the Circle

The global Key determines what pitch is 1/1 in your tuning, which defaults to C. Note names are determined according to a Circle of Fifths having either 12 or 41 fifths. If you change the distance from

C up to A to something other than the default (10,12), then the note names have to adjust. Sounding pitches are always displayed both as a 41ET note name (in column 7), and as a 12ET MIDI note name (in the rightmost 2 columns). To see how all of these parameters work together, for example, your 1/1 could be the pitch A, and you want the interval from 1/1 up to A to be 5/3. To do this, first type in 5/3 in the *from 1/1 up to A* = field, and then type 5/3 in the Tuning Formula field.



A4 = 440 Hz

from 1/1 up to A = 5/3

☐ don't consider in MIDI OUT +/- cents values

Letter: A

Accidental: ♮

Set the Key by choosing a letter name ...

A
5/3

Circle has ... ○ 41 Fifths [✓ 12 Fifths]

... or set the Key from a Tuning Formula

5/3

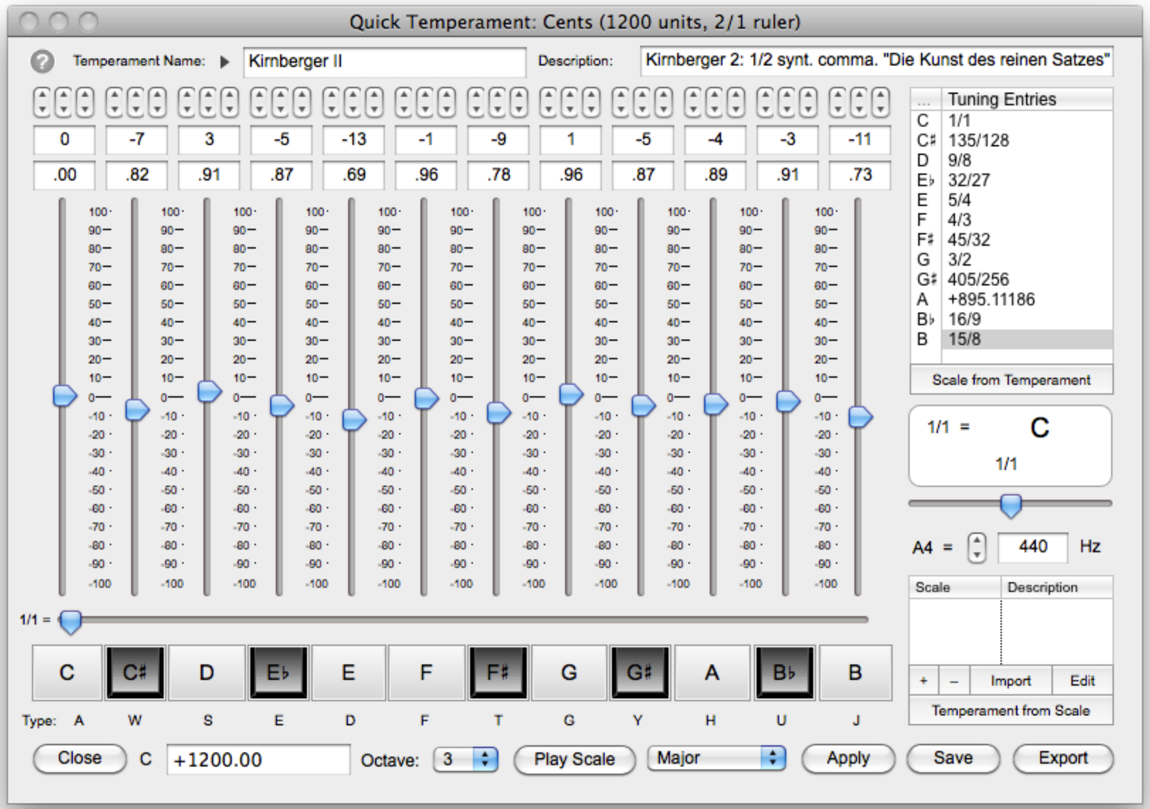
☐ Hz Mode (bypass Key) ☐ Force entries to Hz

Since you have chosen to define the interval from 1/1 up to A as 5/3, the tone 6/5 will be not be C, but will be +C. If you wanted the tone 6/5 to be notated as C instead of +C, then from 1/1 up to A should not be 5/3, but instead the default (10,12) or the 41ET version (32,41). If choose either of these values, make sure that you choose the corresponding 'Circle has ...' 12 or 41 fifths radio-button.

By default the Key is C, number of fifths on the circle is 12, and the position of A is (10,12), so the tuning entry 1/1 results in a sounding pitch of C ± 0¢. If the position of A is changed to something other than (10,12), then the sounding pitch for C will not be C ± 0¢. For example, if A is (32,41), then in the Key of C, 1/1 will result in a sounding pitch of C -7.3¢, because A (32,41) is 7.3¢ higher than A (10,12), and the Hz value of A4 is fixed to the same frequency in both situations.

If you work exclusively in Hz, you may want to bypass the Key settings altogether by selecting *Hz Mode*. As a convenience for those inputing only Hz entries, there is also an option to *Force entries to Hz*.

7. Quick Temperaments



Quick Temperament: Cents (1200 units, 2/1 ruler)

Temperament Name: Kirnberger II

Description: Kirnberger 2: 1/2 synt. comma. "Die Kunst des reinen Satzes"

C	C♯	D	E♭	E	F	F♯	G	G♯	A	B♭	B
0	-7	3	-5	-13	-1	-9	1	-5	-4	-3	-11
.00	.82	.91	.87	.69	.96	.78	.96	.87	.89	.91	.73
A	W	S	E	D	F	T	G	Y	H	U	J

...	Tuning Entries
C	1/1
C♯	135/128
D	9/8
E♭	32/27
E	5/4
F	4/3
F♯	45/32
G	3/2
G♯	405/256
A	+895.11186
B♭	16/9
B	15/8

1/1 = C 1/1

A4 = 440 Hz

C +1200.00 Octave: 3 Major

Quick Temperaments are 12-tone-per-octave tunings which are applied to the entire range of the currently selected tuning layer of the currently selected keyboard. This feature is new in version 2.0, and was added at customer request.

Faders

The vertical faders control the pitch of each key within a 12-tone temperament octave, + or - an equal tempered halfstep. The horizontal 1/1 fader below the vertical faders controls which key is assigned to the root (0.00 unit or 1/1) of the temperament. The global Hertz value of A4 can be set with the small horizontal slider, affecting the tuning of all pitches. Note that the global Key and Hertz settings control the sounding pitch of 0.00 units or 1/1 of all tunings in CSE.

Little Up/Down Arrows

The triple arrow controls above each vertical fader change the unit value of each fader in increments of whole, tenths, and hundredths of a unit respectively. The global Hertz value of A4 can also be set using arrows.

Buttons

Click the buttons below the faders to hear the pitches of each key. Typing on the keyboard with the indicated keys (roughly in the pattern of a traditional piano keyboard) also plays the pitches of the temperament.

Entry Fields

The temperament name and description can be edited in entry fields. Use the entry field below the key buttons to tune keys using tuning entries, as is done in the main window of CSE.

Lists

The Tuning Entries listbox shows the values for each key. The entries may be edited directly by double clicking, and single or multiple keys may be edited using the field at the bottom of the window. The Scale listbox is identical to the scale listbox in the main window of CSE. Any scales which have been imported will appear there, and any changes made to the list will appear in the main window when the temperament window is closed.

Temperaments and Scales

Imported scales are in the Scala file format. CSE Temperaments have their own file types. The Scale from Temperament button below the Tuning Entries listbox can be used to create a Scala file from a temperament. Likewise, the Temperament from Scale button below the scale listbox can be used to create a temperament file from an imported scale. Since imported scales may have a greater or lesser number of pitches in them than 12, smaller numbers of pitches result in temperaments having some degrees left unshifted from 12ET, and larger numbers of pitches result in a dialog allowing you to choose which pitches you want to use as a subset of the scale.

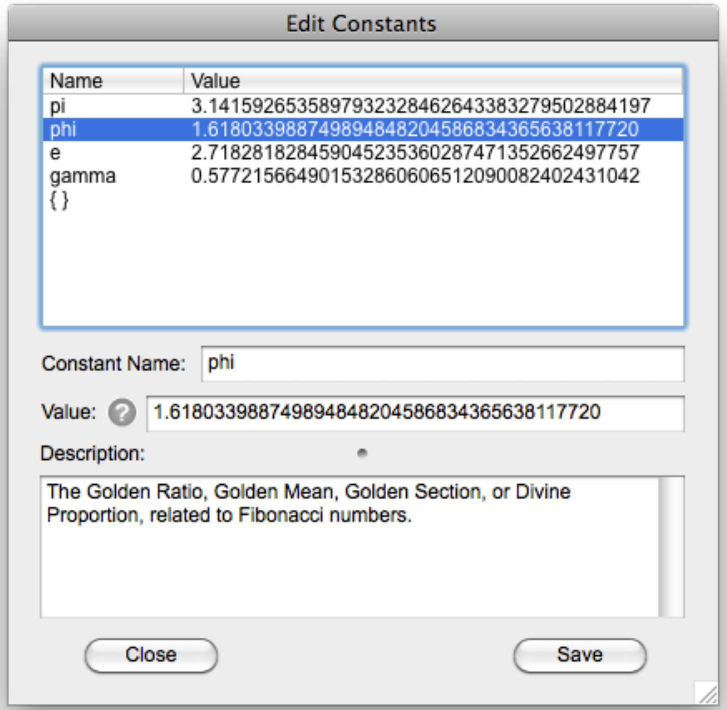
Popup Menus

The current Temperament being edited can be chosen by clicking the disclosure triangle next to the temperament name field. The sounding octave can be changed using the Octave popup menu. Scales can be auditioned by selecting a scale type and pressing the Play Scale button. Pitches from scales which have been imported are accessible by clicking disclosure triangles in the scale list.

Exporting

Temperaments can be exported in a variety of file formats including Scala .scl, Anamark .tun, Native Instruments Pro53 .p5m, Cubase / Nuendo .xml, and Max Magic Microtuner .mtx Keep in mind that some of these formats are not available from the CSE main window, and some that are available there are not available from the Quick Temperament window, because temperament files are limited to 12 tones. For example, Kontakt .txt script exporting is not available from the Quick Temperament window, but it is possible to export the temperament to a Kintakt script by first applying the temperament across the keyboard, then closing the Quick Temperaments window, and exporting the tuning from the CSE main window. Temperament files are stored in the CSE Application Support folder.

8. Constants



Name	Value
pi	3.1415926535897932328462643383279502884197
phi	1.618033988749894848204586834365638117720
e	2.718281828459045235360287471352662497757
gamma	0.577215664901532860606512090082402431042
{ }	

Constant Name: phi

Value: 1.618033988749894848204586834365638117720

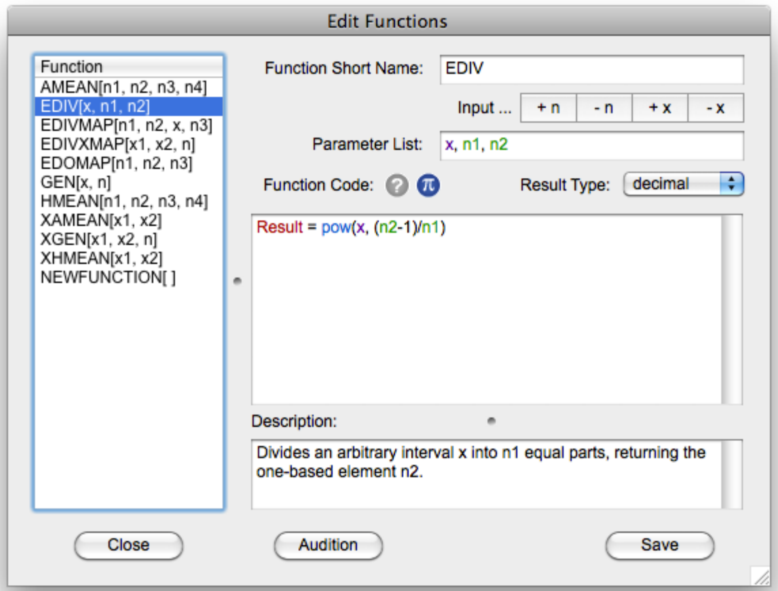
Description:

The Golden Ratio, Golden Mean, Golden Section, or Divine Proportion, related to Fibonacci numbers.

Open the Constants window by selecting View > Constants. Constants are named double precision values. A constant name consists of lowercase letters only with no spaces, surrounded by curly brackets, as in {yourconstant}, and a constant value may be any positive number between 0 and 1.7976931348623157e+308.

Note that scientific notation is not recognized. Constants can be used in Tuning Formulas, such as {pi}/(3/2), which would be the interval 3.14159... transposed down by the interval 3/2. Constants can also appear in Functions and Algorithms. For more information, see the Function and Algorithm editors.

9. Functions



Function
AMEAN[n1, n2, n3, n4]
EDIV[x, n1, n2]
EDIVMAP[n1, n2, x, n3]
EDIVXMAP[x1, x2, n]
EDOMAP[n1, n2, n3]
GEN[x, n]
HMEAN[n1, n2, n3, n4]
XAMEAN[x1, x2]
XGEN[x1, x2, n]
XHMEAN[x1, x2]
NEWFUNCTION[ ]

Function Short Name: EDIV

Parameter List: x, n1, n2

Function Code:

Result Type: decimal

```
Result = pow(x, (n2-1)/n1)
```

Description:

Divides an arbitrary interval x into n1 equal parts, returning the one-based element n2.

Open the Functions window by selecting View > Functions. Functions are user programmable single key tuning structures which perform computations on any number of input values to return a single output value. The input values (Parameters) can be positive integers (whole numbers) or decimals (double precision numbers), and the output value (Result) may be an integer or a decimal.

Functions are named using uppercase letters only with no spaces, such as YOURFUNCTION, followed by a comma separated list of parameters enclosed in brackets, such as [n1, n2, x1, x2] called a Parameter List. Note that brackets distinguish Functions from other tuning operations which use parentheses. The name and parameter list together designate a function. The example would appear as YOURFUNCTION[n1, n2, x1, x2]. Each n in the Parameter List is an integer, and each x is a decimal. You assign the order and the data type of each parameter in the list, and you determine how the parameters are used to compute the output using the Function Editor.

Function instructions (entered in the Function Code editing field) must conform to the list of BASIC programming language structures outlined in the Function Help hierarchical popup menu. Rudimentary knowledge of computer programming is helpful; non-programmers should study the examples and learn by exploring. In order to work properly, the function must set the value of Result, with Result = ..., for example:

```
Result = n1/n2
```

Constants can be used in the function code, using the Constants popup menu. See the Constants Editor for more information.

Functions may be tested using Audition, which provides dynamic control of each variable in real time. The results of Auditioning a function may be appended to a list, copied to the Scratchpad or Clipboard, and pasted into your tuning.

Once a function is finished, it is usable as part of any Tuning Formula. Functions are parsed independently and may be combined with other tuning operations. For example, the function EDIV[x, n1, n2] may be used in a tuning entry such as (9/8)*(3,17) to create something like:

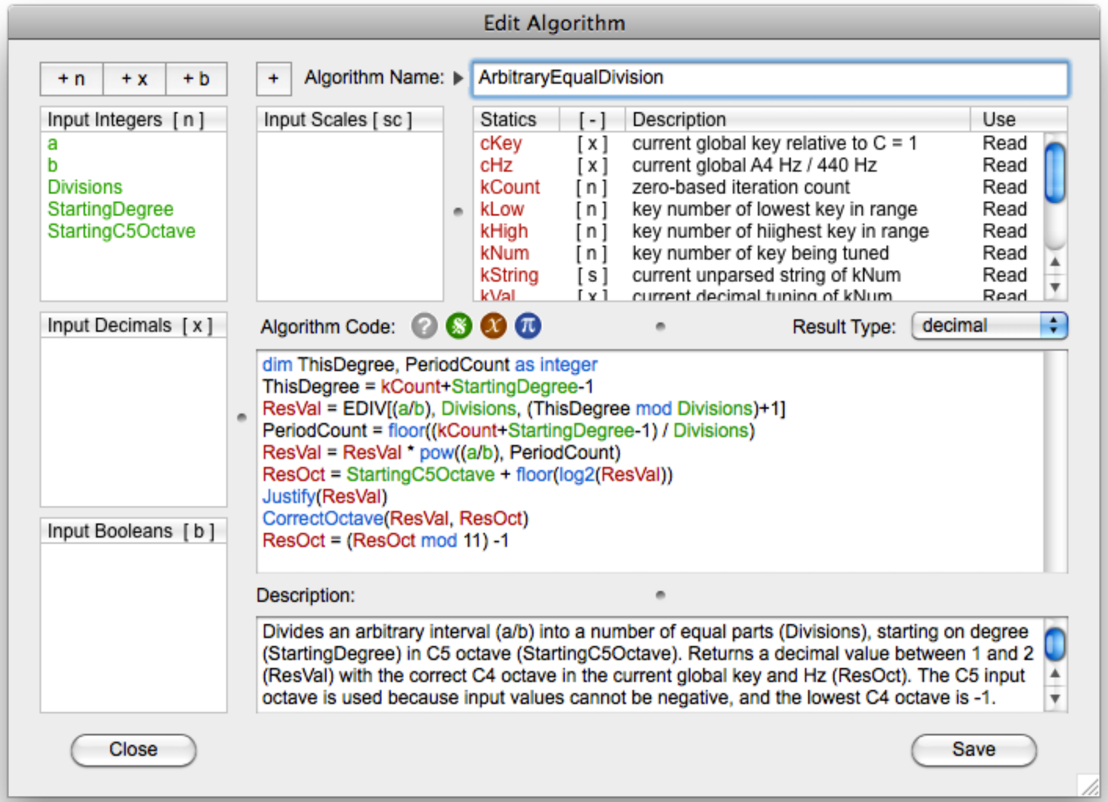
(9/8)*EDIV[3, 13, 4]*(3,17)

This would be (9/8) transposed up by the 4th degree of an equal division of 3 into 13 parts (the Bohlen-Pierce scale), transposed up again by the 3rd degree of 17ET. Further, any function variable which is an x (a decimal) may also be entered as a tuning operation itself, such as EDIV[(13/8), 5, 2]. This would be the 2nd degree of an equal division of the interval 13/8 into 5 parts. Constants may also be entered as variables, using the Constants popup menu, giving possibilities such as EDIV[{pi}, 12, 7], which would be the 7th degree of an equal division of the interval pi (3.14159...) into 12 parts. See the Constants Editor for more information.

Functions may be used iteratively and their functionality further expanded using higher level structures called Algorithms.

Function files are stored in the CSE Application Support folder.

10. Algorithms



Algorithm Name: ArbitraryEqualDivision

Input Integers [n]
a
b
Divisions
StartingDegree
StartingC5Octave

Input Scales [sc]

Input Decimals [x]

Input Booleans [b]

Statics	[-]	Description	Use
cKey	[x]	current global key relative to C = 1	Read
cHz	[x]	current global A4 Hz / 440 Hz	Read
kCount	[n]	zero-based iteration count	Read
kLow	[n]	key number of lowest key in range	Read
kHigh	[n]	key number of hiighest key in range	Read
kNum	[n]	key number of key being tuned	Read
kString	[s]	current unparsed string of kNum	Read
kVal	[x]	current decimal tuning of kNum	Read

Algorithm Code:

Result Type: decimal

```
dim ThisDegree, PeriodCount as integer
ThisDegree = kCount+StartingDegree-1
ResVal = EDIV[(a/b), Divisions, (ThisDegree mod Divisions)+1]
PeriodCount = floor((kCount+StartingDegree-1) / Divisions)
ResVal = ResVal * pow((a/b), PeriodCount)
ResOct = StartingC5Octave + floor(log2(ResVal))
Justify(ResVal)
CorrectOctave(ResVal, ResOct)
ResOct = (ResOct mod 11) -1
```

Description:

Divides an arbitrary interval (a/b) into a number of equal parts (Divisions), starting on degree (StartingDegree) in C5 octave (StartingC5Octave). Returns a decimal value between 1 and 2 (ResVal) with the correct C4 octave in the current global key and Hz (ResOct). The C5 input octave is used because input values cannot be negative, and the lowest C4 octave is -1.

Open the Algorithms window by selecting View > Algorithms. Algorithms are user programmable tuning structures which perform computations on any number of input values to return multiple output values for a range of keys. The input values (Parameters) can be positive integers (whole numbers), decimals (double precision numbers), or boolean values (true or false). After input parameters are passed to an algorithm, the algorithm may be called any number of times (iterations) using those input parameters, so that a range of keys may be tuned through a process.

Algorithms are named using both uppercase and lowercase letters, as in YourAlgorithm, followed by a comma separated list of parameters, named with any combination of letters and numbers, enclosed in angle brackets, such as <FirstParameter, SecondParameter, Your3rdParameter>. You assign the names and data types of each parameter, but the order of their appearance in the parameter list is automatic; from left to right, the order is first the integers, then the decimals, and lastly the Booleans.

To give multiple unique results, a variety of static values (Statics) are available for algorithm computations. Two decimal statics provide global information about the current key (cKey) and current

A4 Hertz (cHz). The static cKey is a ratio of the current key to C where C = 1. The static cHz is a ratio of the current A4 Hz value to A4 = 440. For example, you may have chosen a key of D = 9/8 and A4 = 445 Hz. In this case, cKey will be 9/8 = 1.125 and cHz will be 445/440 = 1.0113636...

For each iteration, there are two output values: the tone value (ResVal) which may be either a decimal or a string, and the C4 octave number (ResOct), which should be an integer between -1 and 9. ResVal as a string may be used to produce results including ratios or functions or any combination of normal tuning entries which can then be parsed as any other tuning entry. ResVal as a decimal will create tuning entries which are already decimals and do not need to be parsed.

There are two main statics which are used to make the results of each iteration unique. The first is called kCount, which keeps track of iterations beginning with 0 and incrementing with each iteration, and the other is called kNum, which is the actual key number of the key which will receive the results of an iteration. kLow and kHigh are the numbers of the lowest and highest keys in the currently selected range, which can be used to determine the size as well as the register of the range. Statics related to kNum are kString, kVal and kOct, useful for computing transformations in which the current string, decimal or octave values of a key are used in some way to determine the output value for that key.

When ResVal returns a decimal, it should have a value 1 <= ResVal > 2, meaning its value is within an octave (2/1). To ensure this range of values for ResVal or any other decimal you work with, you can use the built-in tuning function Justify(x). If you are working with ratios, you can also use the function JustifyRatio(n1, n2) where the ratio is in the form n1/n2 where n1 > n2. Other useful tuning functions convert decimals to cents, or cents to decimals, and instead of changing the value passed, return new values so that you can assign one variable to the conversion of another, as in:

```
FirstParameter = 1.6180339887498948482045868343656381177720
SecondParameter = 88 + DecimalToCents(FirstParameter)
Your3rdParameter = (3/2) * CentsToDecimal(SecondParameter)
```

Assigning C4 octave numbers can be a somewhat tricky business, because the values of cKey and cHz change the octave break within a decimal based octave. To simplify the process of assigning C4 octaves to ResOct, within the algorithm octave numbers may be calculated as if in the key of C and A4=440 Hz (so that cKey=1 and cHz=1). Then, before returning ResOct, you may use the tuning function CorrectOctave(ResOct) to automatically correct the octave number according to the current key and Hertz.

When ResVal returns a string, the string has to be in valid entry format; that is, it must be able to be parsed in the same way that a tuning entry is parsed. Since string manipulation is not the same as numerical calculation, computations should be performed using kVal, and you may need to introduce other variables into the code in order to work effectively with both kString and kVal. For an example of how to work with both strings and decimals, see the PrimeLimit, OddLimit, and OddHarmonics algorithms, which also show how to sort string arrays to match the order of an array of decimals.

Algorithm instructions (entered in the Algorithm Code editing field) must conform to the list of BASIC programming language structures outlined in the Algorithm Help hierarchical popup menu. Rudimentary knowledge of computer programming is helpful; non-programmers should study the examples and learn by experimentation. In order to work properly, the algorithm code must somewhere set the values of ResVal and ResOct, using ResVal = ... , and ResOct = ... , for example:

```
ResVal = kVal * (FirstParameter / Your3rdParameter) * cKey * cHz
ResOct = floor(log2(ResVal))
```

Algorithm code may employ Functions and Constants using the popup menus in the editor. Algorithm parameters, statics and constants may be passed to Functions within the algorithm code. See the Function and Constant Editors for more information.

Algorithm code may also reference imported scales, using scX(sc, n) or scS(sc, n). The Sign icon gives a list of scales in the editor. This should be used with caution, because if the algorithm is called but the scale referenced has not been imported, an error will occur.

When working with tuning entries formatted as Hz values, algorithms the return type of the algorithm must be a string, and the string must begin with the characters f= ... What follows after the equals is a Hz value. Hz frequency entries require that the decimal value and C4 octave number be calculated from the frequency. The tuning methods HzEntry(x) and HzOctave(x) should be used for this purpose. For example, the following code formats a result value as a correct frequency entry, and returns the correct C4 octave number, where x is a Hz value:

```
ResVal = “f=” + str(HzEntry(x))
ResOct = HzOctave(x)
```

Algorithms may be tested (outside of the editor) using Audition, which returns results in a list. The results may be copied to the Scratchpad or Clipboard, and pasted into your tuning. It is possible that a coding error in your algorithm may cause the computer to hang or crash, but most errors are caught and displayed at runtime, to help you debug your code.

Algorithm files are stored in the CSE Application Support folder.

Credits

All versions of CSE are designed and programmed by Aaron Andrew Hunt, using Xojo and MBS Plugins on a Mac.

This documentation was written by Aaron Andrew Hunt, using Apple Pages.

Thank you for supporting H-Pi Instruments.

APPENDIX

Language Reference

The following relates to user programmable Functions, Algorithms, and entry Code Snippets. Below, an n represents an integer (whole number), an x represents a double precision (decimal) number, a b represents a Boolean (true or false) value, an s represents a string (text), and a v represents any of these types.

Operators:

+	Addition
-	Subtraction
*	Multiplication
/	Division
n1 Mod n2	Modular arithmetic, the remainder of the division of n1 by n2.
<	Less than
=	Equals
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to
And	Boolean AND
Not	Boolean NOT
Or	Boolean OR

Comments:

‘

//

REM

Data Types:

Dim YourVariableName ...	Dimension a variable, including arrays
Redim YourVariableName ...	Redimension a variable, including arrays
as Integer	YourVariable is a whole number = n
as Single	YourVariable is a single precision decimal number = x
as Double	YourVariable is a double precision decimal number = x
as Boolean	YourVariable is a boolean (true or false) value = b
as String	YourVariable is a string (text) = s

Arrays:

YourArray(n)	An array of n number of values, dimensioned as any data type. Multidimensional arrays are YourArray(n1, n2 ...)
YourArray.append(v)	Places a new element with value v at the end of an array. The data type of v must agree with YourArray's data type.
YourArray.insert(n, v)	Places a new element at the index n of an array. Previous element n becomes element n+1. The data type of v must agree with YourArray's data type.
v = YourArray.pop	Removes last element from YourArray and returns its value. The data type of v must agree with YourArray's data type.
YourArray.remove(n)	Removes element n from YourArray.
YourArray.Sort	Sorts YourArray in ascending order (one dimensional only).
YourArray.SortWith(Array1, Array2, ...)	Sorts Array1, Array2 ... in the same order as YourArray.
n = YourArray.IndexOf(v)	Returns the index n of the element having the value v. The data type of v must agree with YourArray's data type.
n = Ubound(YourArray)	Returns the number of elements in the array -1.

Control Structures:

Function... — A Method which returns a value.

```
Function YourMethod(v1 as DataType, v2 as DataType ... ) as DataType
    ...
Return v
```

Sub... — A Method which does not return a value.

```
Sub YourMethod(v1 as DataType, v2 as DataType ... )
    ...
Return
```

For... Next — A loop. Replace 0 and 127 with any integers in ascending order. Use DownTo in place of To for descending loops.

```
For n = 0 To 127
    ...
Next n
```

Exit — Exits a loop.

If... Then... End If — Execute code blocks conditionally. Replace v1 = v2 with any condition.

```
If v1 = v2 Then
```

```
        ...
    End If
```

If... Then... Else... End If	Execute two code blocks conditionally. Replace v1 = v2 with any condition.

```
    If v1 = v2 Then
        ...
    Else
        ...
    End If
```

If... Then... ElseIf... Then... Else... End If

```
    If v1 = v2 Then
        ...
    ElseIf v1 = v3 Then
        ...
    Else
        ...
    End If
```

Execute any number of code blocks conditionally.

Replace v1 = v2 and v1 = v3 with any conditions.

Do... Loop	Repeatedly execute a code block until a condition is met. Replace v1 = v2 with any condition.

```
    Do Until v1 = v2
        ...
    Loop

    Do
        ...
    Loop Until v1 = v2
```

Condition can be checked at the start or the end of the loop.

Select Case... End Select	Execute code blocks based on a variable value.

```
    Select Case v1
    Case v2
        ...
    Case v3
        ...
    End Select
```

When v1 = v2, execute ...

When v1 = v3, execute ...

While... Wend	Repeatedly execute a code block while a condition is met. Replace v1 = v2 with any condition.

```
    While v1 = v2
        ...
    Wend
```

Return	Go back to the calling method.
Return v1	Return the value v1 back to the calling method.

Numbers:

x = Abs(x)	Absolute value of x, forcing negative to positive.
x = Acos(x)	Arccosine of x. The angle with the cosine value x in radians.
x = Asin(x)	Arcsine of x. The angle with the sine value x in radians.
x = Atan(x)	Arctanget of x. The angle with the tangent value x in radians.
x = Atan2(x1, x2)	Arctanget of the point x1, x2, where x1 = x, and x2 = y.
x = Ceil(x)	Ceiling returns the value x rounded up to the nearest integer.
x = Cos(x)	Cosine of x, in radians.
x = Exp(x)	The constant *e* raised to the value x.
n = FindGCD(n1, n2 ...)	Greatest Common Divisor of a set of integers.
n = FindLCM(n1, n2 ...)	Lowest Common Multiple of a set of integers.
x = Floor(x)	The value x rounded down to the nearest integer.
b = isNANorINF(x)	Returns true if x is infinity or if x is not a number.
x = Log(x)	The base *e* logarithm of x.
x = Log2(x)	The base 2 logarithm of x.
x = Max(x1, x2 ...)	The largest value from a set of numbers.
x = Min(x1, x2 ...)	The smallest value from a set of numbers.
x = Pow(x1, x2)	The number x1 raised to the power of x2.
x = Rnd	Random value between 0 and 1.
x = Round(x)	The value x rounded to the nearest integer.
x = Sin(x)	Sine of x, in radians.
x = Sqrt(x)	The square root of x.
x = Tan(x)	Tangent of x, in radians.

Strings:

n = Asc(s)	The ASCII value of the first character of a string.
x = CDbl(s)	Convert a string to a number, using local decimal separator.
s = Chr(n)	The character whose ASCII value is n.
n = CountFields(s1, s2)	The number of fields in s1 using s2 as a separator.
s = Format(x, s)	The number x formatted according to the string s.
s = Hex(n)	Hexadecimal equivalent of n.
n = InStr(n, s1, s2)	Begin at character n and look for s2 within s1.
s = Left(s, n)	The left part of string s, n characters in length.
n = Len(s)	The length of a string.
s = Lowercase(s)	Force a string to lowercase.
s = LTrim(s)	Remove leading spaces from a string.
s = Mid(s, n1, n2)	A part of string s, n2 characters in length, starting at n1.

s = NthField(s1, s2, n)	Field number n in string s1 using s2 as a separator.
s = Oct(n)	The octal equivalent of n.
s = Replace(s1, s2, s3)	In the string s1, replace the first occurrence of s2 with s3.
s = ReplaceAll(s1, s2, s3)	In the string s1, replace all occurrences of s2 with s3.
s = Right(s, n)	The right part of string s, n characters in length.
s = RTrim(s)	Remove trailing spaces from a string.
s = Str(x)	Convert a number to a string.
n = StrComp(s1, s2, n)	Compare strings n = 0 = case-sensitive, 1 = lexicographic. s1 < s2 = returns -1, s1 = s2 returns 0, s1 > s2 returns 1.
s = Titlecase(s)	Force a string to Titlecase (first character uppercase).
s = Trim(s)	Remove leading and trailing spaces from a string.
s = Uppercase(s)	Force a string to all UPPERCASE characters.
x = Val(s)	Convert a string to a number using a decimal point as the separator character.

Tuning:

CorrectOctave(s, n)	Fixes n to the correct C4 octave, where s is a tuning entry.
CorrectOctave(x, n)	Fixes n to the correct C4 octave, where x is a tuning entry.
Justify(x)	Places the decimal value of x within octave boundaries 1 <= x < 2
JustifyCents(x)	Places the cent value of x within octave boundaries 0 <= x < 1199.99
JustifyRatio(n1, n2)	Places a ratio n1/n2 within octave boundaries 1/1 <= n1/n2 < 2/1
x = DecimalToCents(x)	Returns a cent value (where an octave = 1200) from a decimal value (where an octave = 2.0).
x = CentsToDecimal(x)	Returns a decimal value from a cent value.
x = HzEntry(s)	Returns a valid tuning entry decimal from a Hz value input as a string. NOTE: the string can be a CodeSnippet inside <>
x = HzEntry(x)	Returns a valid tuning entry decimal from a Hz value input as a decimal.
n = HzOctave(x)	Returns the correct C4 octave number from a Hz value input as a decimal.
x = entryToX(s)	This is actually the parsing engine itself, so it allows nested tuning entries (where s is a tuning entry).
s = scS(sc, n)	Tone n from imported scale sc in the form of a string value.
x = scX(sc, n)	Tone n from imported scale sc in the form of a decimal value.
n = scLen(sc)	The number of tones in an imported scale sc.