

OWT: Real-time software for exploring optimal tuning systems

USER'S MANUAL

Larry Polansky

Dept. of Music, Dartmouth College

Dan Rockmore

Dept. of Math, Dept. of Computer Science, Dartmouth College

Micah K. Johnson

Dept. of Brain and Cognitive Sciences, Massachusetts Institute of
Technology

Douglas Repetto

Computer Music Center, Dept. of Music, Columbia University

Wei Pan

Dept. of Computer Science, Dartmouth College

February 20, 2008

(Rev. 8/22/08)

<http://eamusic.dartmouth.edu/~larry/owt/index.html>

USER'S MANUAL

TABLE OF CONTENTS

	<u>Page #</u>
1.0 GETTING STARTED	1
1.1 Start it!.....	1
1.2 Tuning Settings.....	3
1.3 Understanding the Result	4
1.3.1 The Result View.....	5
1.3.2 The Real-time Player	6
1.4 Using the Built-in Tunings	7
1.5 Save and Load Your Own Tunings	7
1.6 Tunings with fewer than 12 pitches	8

1.0 GETTING STARTED

1.1 Overview

The OWT MIDI Player is a real-time exploration and demonstration tool for the *Optimal Tuning System* project described in our paper “A Mathematical Model for Optimal Tuning Systems,” (forthcoming in *Perspectives of New Music*, available at <http://eamusic.dartmouth.edu/~larry/owt/>). This software demonstrates our constraint-based algorithm for tuning system optimization. The fundamental idea is that a tuning system can be derived via the optimization of an error function defined on several (user-specified) constraints or parameter values (number of pitches, repeat factor, ideal intervals, key and interval weights). A good historically significant example of the relevance of this approach is the construction of well temperaments from the Baroque period. However, the idea is more general, and is applicable to a great many tuning systems throughout history and from around the world (such as Central Javanese slendro).

The program is written in JAVA and runs on both Microsoft Windows and MacOSX. However, we recommend Microsoft Windows for better performance.

Windows:

Double click the *owt.jar* file (assuming the latest release of *JAVA* (<http://java.com>) is installed). The program will first ask you to choose a MIDI device.

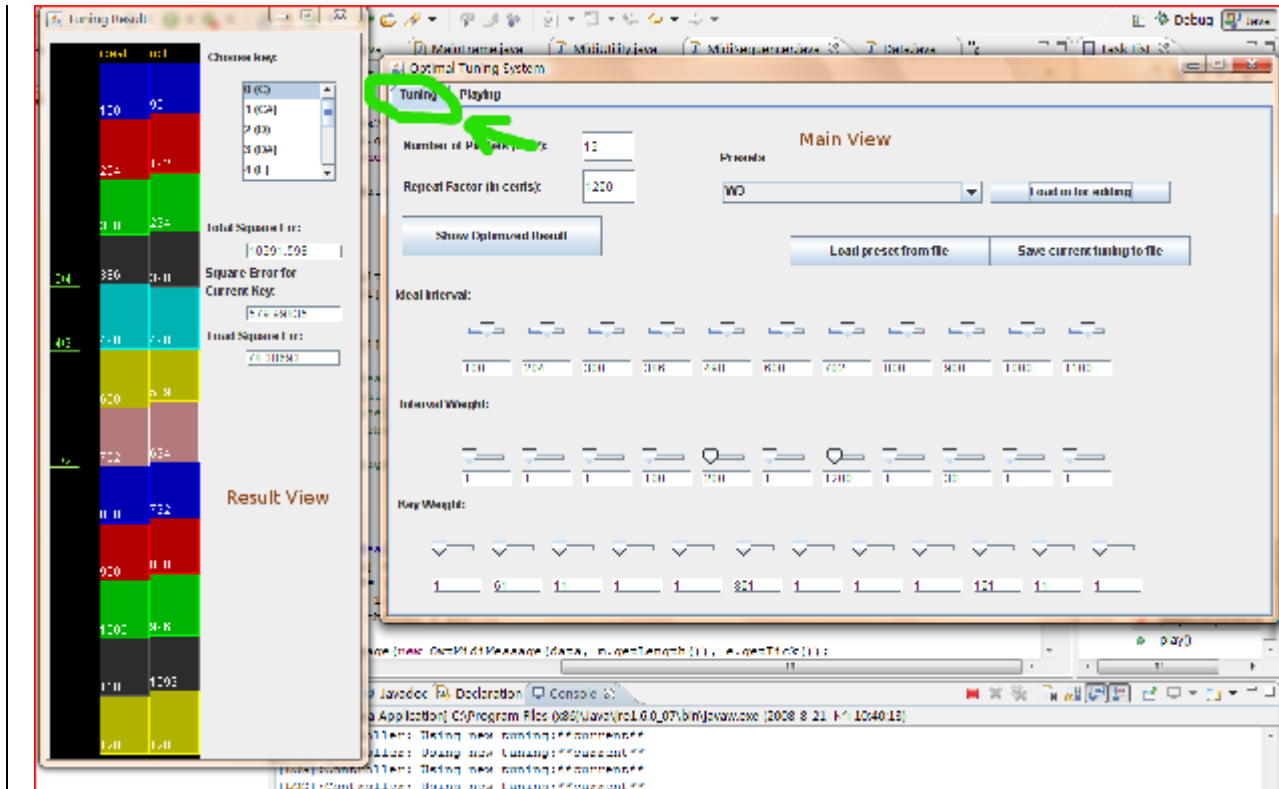


Choose a MIDI device to play MIDI files (recommended: *Microsoft GS SoftSynther* if no other software/hardware synthesizers are available). Choose *OK* to confirm the MIDI device, or *Cancel* to let the software choose a device.

Mac OSX:

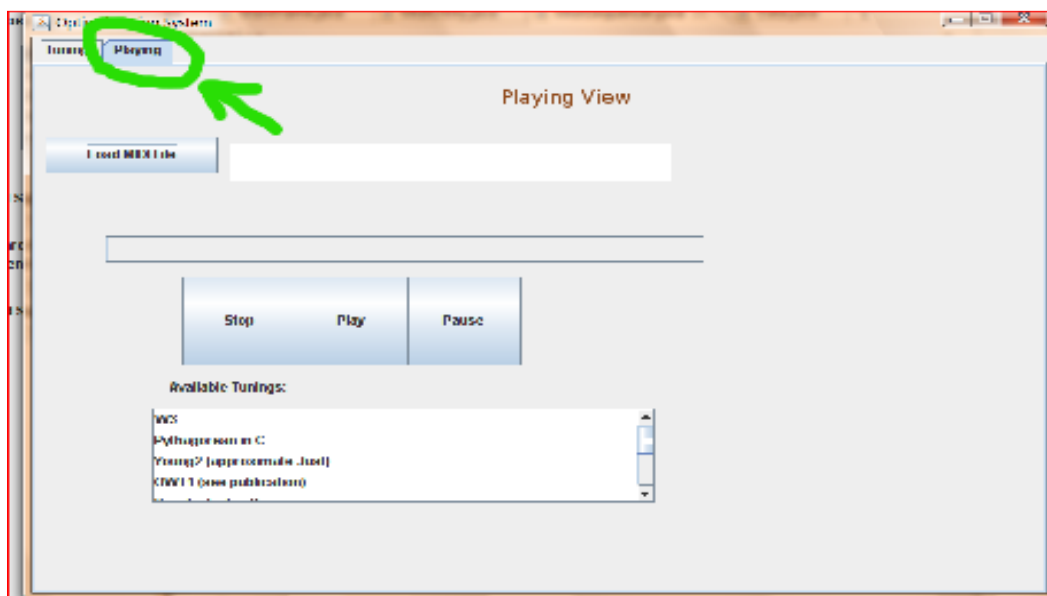
The current version doesn't allow the user to choose a MIDI device for Mac OSX.

Once you start the application, you will see the *tuning* page or *main view*:



The *main view* consists of two sub-windows: *Result View* and *Main View*. (If you don't see the *result view*, click the "Show Optimized Result" button to open it).

To see the *playing* page, or *playing view*, click the *Playing* tab.



1.2 Tuning Settings

As described in our paper (“A Mathematical Model for Optimal Tuning Systems,” forthcoming in *Perspectives of New Music*) an *optimized tuning* is computed from the following user input parameters, representing a number of widely used constraints:¹

- *Repeat Factor*
- *Number of Pitches*
- *Ideal Intervals*
- *Interval Weights*
- *Key Weights*

These are the parameters that can be modified in the main view. The software computes, in real-time, the best possible solution to using an error minimization algorithm.

Repeat Factor and Number of Pitches

The user-specified number of pitches should be between 5-12 and the repeat factor must be that of an octave (1200¢ +/- 50). (Note: this restriction is due to the way in which this software handles arbitrary MIDI files. Our general algorithm has no such constraint — systems may contain any number of pitches and the repeat factor can be anything – e.g., several octaves, an octave and a 5th, etc.).



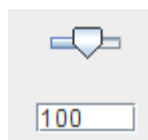
Number of Pitches (5-12):

Repeat Factor (in cents):

When you change the number of pitches, the number of input sliders will change as well. When the number of pitches is changed, the software treats MIDI note numbers differently. See Section 2.1. for details.

Ideal Intervals

The *ideal intervals* for the system are cents values, which represent an ideal difference between scale degrees. For example, in a 12-note system, the 7th ideal interval is the desired value between all P5ths.



Ideal intervals (and weights – see below) are adjusted via sliders or numerical entry. Remember that these values are not the values that will actually be in the derived tuning system, but is a set of values that can be thought of as a target set for all the various intervals (e.g., all P5ths), which is used to construct a matrix and corresponding error function which is minimized for the construction of the actual tuning system. Values are in cents.

¹ We recognize the historical or cultural basis for these constraints and it would be of interest to consider other or additional parameters that might be used to determine a tuning system.

By default, the software displays 11 intervals, representing 12ET. For n pitches there will be $n-1$ interval adjustment boxes.

Ideal Interval:

100 204 300 386 498 600 702 800 900 1000 1100

The first box represents the ideal interval between system degrees separated by 1, the second box by 2, and so on.

Weights (key and interval)

Interval Weight:

1 1 1 1 1 1 1 1 1 1 1

Key Weight:

1 1 1 1 1 1 1 1 1 1 1

Interval weights determine the importance of the corresponding *ideal intervals*. For example, for a typical 12-note European *well temperament*, one might set the interval weights for the P5th and M3rd high.

Key weights correspond to the n pitches in the system, and determine the importance of each “key” in the determination of the error. For example, one might specify, in a well temperament, that the key of $F\#$ is less important than the key of F . One way to think of key and interval weights is as rows and columns of a matrix. The two weights are multiplied together in the optimization algorithm.

1.3 Result View

As values are entered, the software computes an optimal system. Repeat factor and number of pitches cannot be changed in real-time. Ideal intervals, key and interval weights may be changed as a MIDI file is being played, and the tuning that is generated will reflect the optimization of those new parametric values.

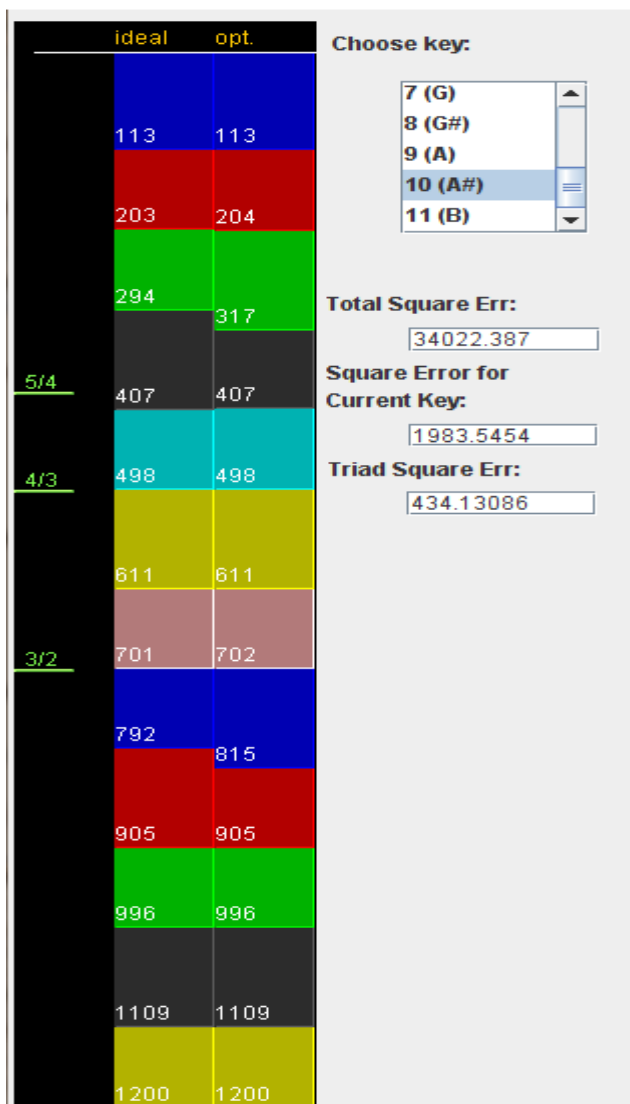
	ideal	opt.	
	113	113	Choose key: <input type="list" value="0 (C)"/> <input type="list" value="1 (C#)"/> <input type="list" value="2 (D)"/> <input type="list" value="3 (D#)"/> <input type="list" value="4 (E)"/>
	203	203	
	294	294	
5/4	407	407	
4/3	498	498	
	611	611	
3/2	701	701	
	792	792	
	905	905	
	996	996	
	1109	1109	
	1200	1200	

Total Square Err:	<input type="text" value="34022.387"/>
Square Error for Current Key:	<input type="text" value="0.7460982"/>
Triad Square Err:	<input type="text" value="437.42572"/>

1.3.1 The Result View

The *Result View* displays the ideal intervals next to the optimally computed intervals. The intervals can be viewed by “key” — as a sequence of intervals, or pitches, starting on a given degree of the tuning system. The key can be selected in the top right. The picture above shows an optimized result in a very highly weighted key of “C” alongside the ideal intervals (one way to achieve a specific “scale” is to set the weight of one key to 1 with all other keys set to 0). As expected, in the above example, the ideal and optimized intervals are the same. (In this case, the ideal intervals are a Pythagorean tuning, whose cents values are rounded off in the display).

However, if we choose the key on the 10th pitch (A#), we see a little bit of deviation between the optimized tuning and the ideal intervals (which are constant):



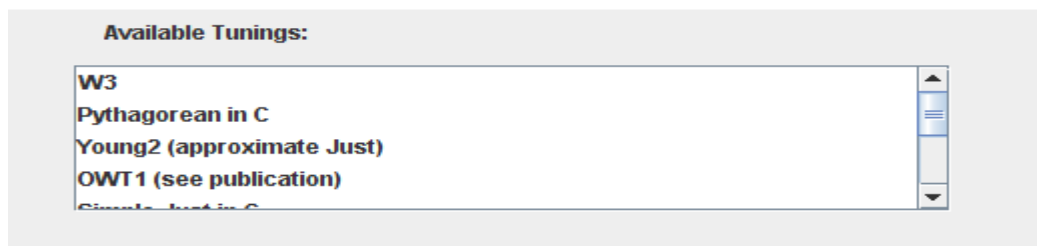
Even for this distant key, the errors are quite small, the $m6^{\text{th}}$ and the $m3^{\text{rd}}$ showing the greatest deviation. The intervals of the $M3^{\text{rd}}$ and $P5^{\text{th}}$, for example, are quite accurate.

Three other (non-editable) displays show the total error for the tuning system (against the ideal interval matrix), the error for that key, and the “triad error”, which uses a measure proposed by Rudolf Rasch (see our paper) which is the mean of the deviation error of triads in the system from simple just triads (the last of these gives some notion of how “good” a well-temperament is, in conventional harmonic terms, according to a very specific metric.).

1.3.2 The Real-time Player

To play a MIDI file, go to the *Playing View* page, and click *Load MIDI*. Next, click the *Play* button. There are also *Stop* and *Pause* buttons

There is a selection box available on this page for built-in tunings (some of them are explained in our paper, and on the website). The user may load in any of those tunings as a place to begin.



The tuning called **current** has a special significance: it specifies that the values (ideal intervals, interval and key weights) displayed in the Main View will be used as input to the optimization. Remember that the user can change the preset tuning by freely altering individual ideal intervals, key and interval weights while the MIDI file is played. The software recomputes the tuning on-the-fly.

1.4 Using Built-in Tunings

Each built-in tuning is simply a set of constraints for optimization: a set of *ideal intervals*, *interval weights*, *key weights* and values for the *number of pitches* and *repeat factor*. The built-in tunings are *not* changeable.

In some cases, like the tuning called *W3* (Werckmeister 3), the optimized result of these values will produce a pre-existing tuning.

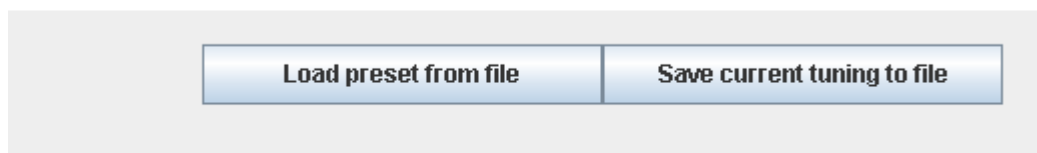
The user can edit the built-in tunings in the *Main View* by choosing one and loading it. This sets all values to those of the built-in tuning. Editing those values will not change the built-in tuning. The user's own tunings may be saved as presets.



Note that if, in *Playing View*, the user selects a tuning other than **current**, the built-in tuning will be generated. Changing values (*interval weights*, *ideal intervals*, etc.) will not affect the tuning unless **current** is set.

1.5 Saving and Loading Tunings

You can store and read easily the your own tunings. The two buttons below in the *Main View* will save the following data: *Repeat Factor*; *Number of Pitches*; *Ideal Intervals*; *Interval* and *Key Weights*.



The data format for saved tunings is pure .txt. The following gives an example:

```
# The file is automatically generated, modification may cause program to halt
# Please read documents before modifying, wei.pan@dartmouth.edu
```

```

# The name of this new tuning
User
# The Repeat Factor (in cent)
1200
# How many intervals
12
# The following three lines are interval, interval weights, and key weights
100 200 300 400 500 600 700 800 900 1000 1100
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0

```

1.6 Tunings with fewer than 12 pitches

As mentioned above, although the algorithm allows for any number of pitches, the software only allows 12 or fewer, for practical reasons having to do with MIDI. With fewer than 12 pitches, the software treats MIDI note numbers differently. Tuning systems must have at least three pitches (C, G, E). When the number of pitches is fewer than 12, “note names” are discarded in the following order:

F# C# G# D# A# B F A D

For example, with an 11-note system, all F#s in the MIDI file will be discarded. For a 10-note system, F# and C# will be discarded. In other words, MIDI note numbers mod 6 and mod 1 should not be used. As a simple example, if a 5-note system is specified, it will be the pentatonic scale. Note that this only pertains to the note-numbers in the MIDI file, it has nothing to do with tuning. With fewer than 12 notes, each note number loses its conventional meaning: the software remaps that note to the new tuning specified by the user.

Revision Sheet

Release No.	Date	Revision Description
Rev. 0	1/20/2008	Initial Version
Rev. 1	2/20/2008	Bug fix and enhanced feature (octave change)
Rev. 2	8/22/2008	Larry's Fix