**NeuroExperimenter Users’ Guide**

[This document was updated 1/13/2022 for version 6.6. ]

This program (NEx) provides an interface to [NeuroSky’s](http://neurosky.com/products-markets/eeg-biosensors/) *MindWave Mobile* (black headset). It should work with other Neurosky hardware, but has not been tested there by me.

Here is the official list of Neurosky devices that *might* work with NEx:

MindWave Mobile (Black headset, *tested with NEx*)

MindWave (RF) (White headset).

MindBand

ThinkCap

TGAM module

TGAT ASIC

BrainLink Pro (a user has reported success in running NEx with this headset).

Note: NeuroSky has discontinued the old interface (SDK) for the new MindWave Mobile 2

headset. Consequently, I have changed NEx to work with the ”Serial Stream protocol”. I believe

some previous versions of NeuroSky headsets will work with the new version of NEx. However

the new interface has been tested successfully with only the MindWave Mobile headsets.

[Neurosky](http://support.neurosky.com/kb) has a knowledge base for all the headsets.

Furthermore, NeuroSky and versions of NEx (after 4.10) of NEx no longer support the white headset. However, the downloads (see below) contain an executable for the white headset: *neuroV4.10xwh.zip*. This is no longer supported by me but should still work. It’s interface differs a bit from that in this document.

The NEx application *does not* require ThinkGear Connector to be running. In fact it should be shutdown in order to run NEx.

Neurosky has a developer [program](http://developer.neurosky.com/). Because NEx does not use the raw data from the headset (just the “power data”), you will have to write your own software if you wish to process that.

Ensure you are running the latest NeuroSky [software](http://developer.neurosky.com/docs/doku.php?id=mindwave).

You can also use NEx with other devices if you can write some simple Windows software: see Appendix 4.

This program only runs under Windows. NEx requires [.NET Framework 4.8](https://dotnet.microsoft.com/download/dotnet-framework/net48) or later and so will *not* run under Windows XP. If you have problems, please consult [Appendix 5](#3j2qqm3).

***Acknowledgement*** : I wish to thank [Vasyl Vernyhora](https://scriptures.ru/yoga/eeg-meditation.htm) for help in testing and for many useful

suggestions.

# Introduction

The purpose of NEx is to explore brainwave output as you attempt different “mind states”, such

as meditation, relaxation, concentration, etc. You can discover which combination of brainwaves

characterize a mind state. You can train yourself to generate that state via visual and audio

feedback. NEx has been used by high school students in science fair projects, college students

and professors in their research, and by therapists investigating the efficacy of neurofeedback

To learn more about brainwaves, EEG’s, and neurofeedback consult the internet. Here are some (of many) resources: [Neurofeedback](http://en.wikipedia.org/wiki/Neurofeedback), [Electroencephalography](http://en.wikipedia.org/wiki/Electroencephalography), [Alpha waves](http://en.wikipedia.org/wiki/Alpha_wave), [Experimental Procedure](http://www.psych.westminster.edu/psybio/BN/Labs/Brainwaves.htm), There are some [caveats](https://www.nytimes.com/2022/01/12/well/mind/neurofeedback-therapy-mental-health.html?action=click&module=Well&pgtype=Homepage&section=Well). There is a [YouTube video](https://www.youtube.com/watch?v=exoeVmGphI0&feature=youtu.be) describing this software.

# Installation

The NEx executable(s) are found [here](https://sites.google.com/view/fredm/home).

For *black headsets*, just download *NeuroExVx.x.zip* . Unzip it in a folder of your choice (probably **not** *C:/Program Files* as you likely do not have permissions there).

The executable for the *white headset* (*neuroV4.10xwh.zip*) is also installed by unzipping it to a folder of your choice.

To uninstall any of these, just delete the folder you unzipped it to. They do not update the registry.

You can ignore any security messages from Microsoft or your virus checker during the

download and installation.

# Quick Start

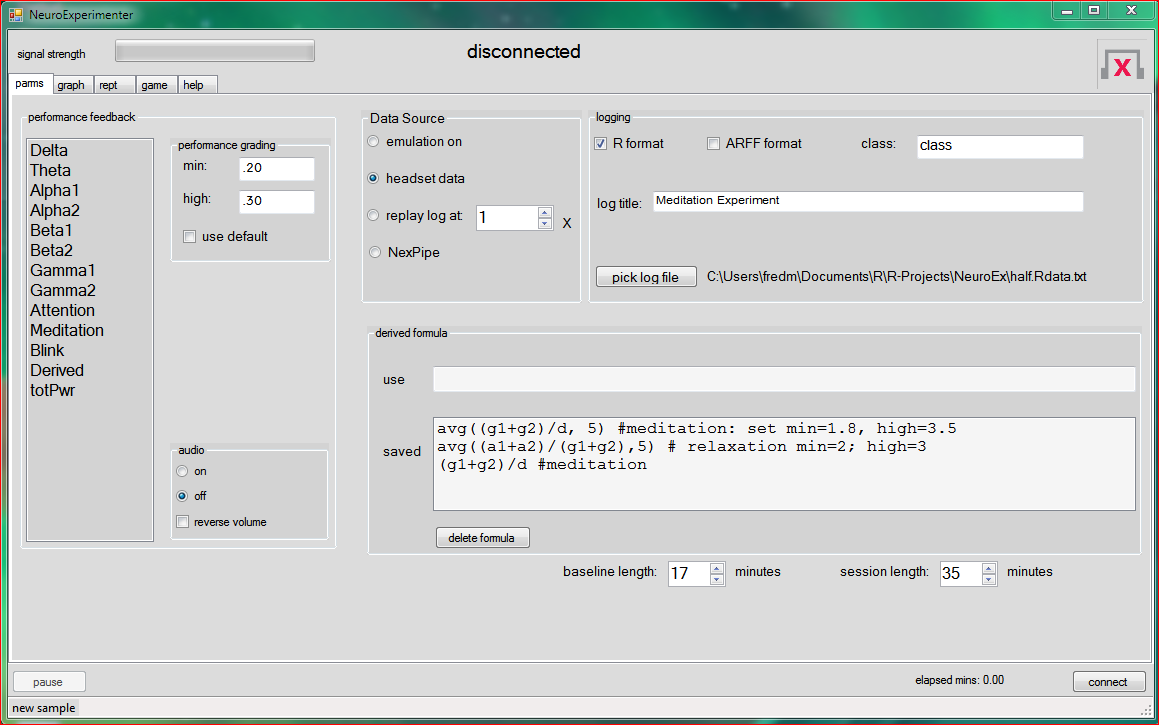
1. Note: you can run NEx without a headset: it has an emulation mode.
2. The installer should have put a shortcut on your desktop. Click on that. If you prefer, you can navigate to the folder where you have installed NEx and click on EEG.exe.
3. The program starts on the *parm* page.
4. Click on *EEG.exe* and the program should run, starting on the *parm* page.
5. Click on *headset data* in the *data source* box.
6. Clear the checkmark in the *R format* box in the *logging* box.
7. Fit the headset to your head; turn it on.
8. Click the *connect* button; NEx will switch to the *graph* page and try to connect to the headset..
9. Click the *Alpha1* button on the *graph* page; watch and listen as your brain generates alpha waves.
10. See Appendix 5 if you have problems.

# Illustrative Example

To introduce most of the features of NEx, I present an actual experiment. There is a YouTube video describing this [here](https://www.youtube.com/watch?v=exoeVmGphI0). Other experiments are described in Appendix 3.

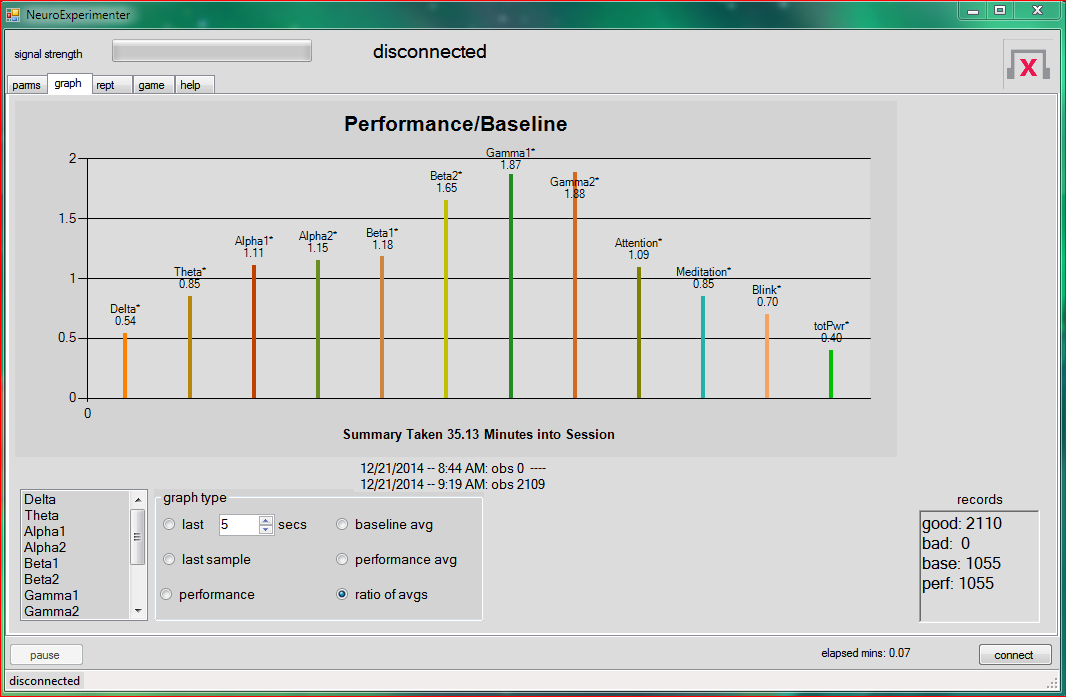
I wanted to determine the difference between meditation and an ordinary mind state. I enlisted the help of “Sally” (not her real name) who was trained by experts and has practiced daily meditation for over 20 years. The following screen shots contain the actual data involved in the experiment.

**Setup**:



We divided the session into 2 parts: baseline (length 17 minutes) and the “performance” section which is the remainder (length 35 - 17 = 18 minutes). This is setup with the *length* *timers* at the bottom. We want to create a log in R format so that box was checked; we let its path default.

We clicked on the *connect* button to start the session, after the headset was fitted and turned on. Sally read a little, surfed the internet, and ate a snack for the first 17 minutes (the “baseline”). When the baseline ended, the system sounded a chime. Then Sally began her meditation. When a total session time of 35 minutes elapsed another chime was sounded. Then the headset was removed and we switched to the graph page to analyze the data. It looked like:

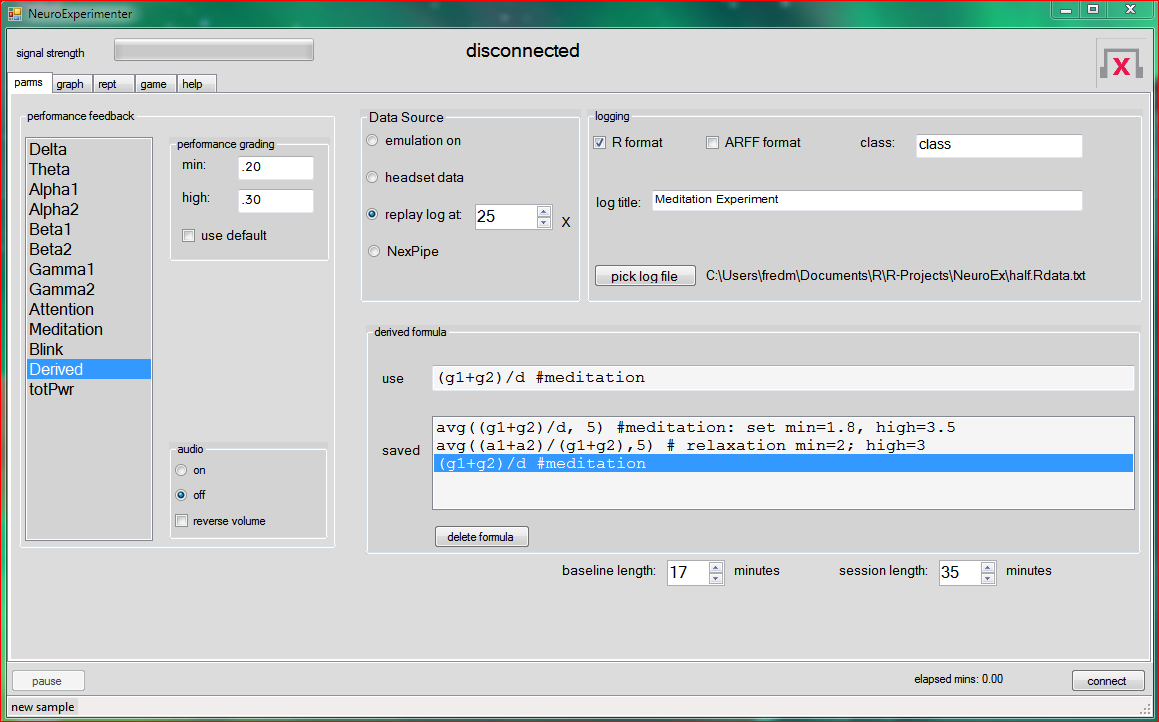


Each bar in this summary represents a ratio of the performance over the baseline, for that wave type. The ratio uses the average output in the waveform (as a percent of total “power” output). For example, the value of 1.09 for Attention means that the average output for Attention in the “performance” part of the session did not differ much from that in the baseline part of the session.

From this chart we can make the following observations:

1. Average Gamma1 and Gamma2 output was significantly higher during mediation (as a percent of total power output). Delta was significantly lower. This suggests that the formula (Gamma1 + Gamma2)/Delta could be used to discriminate between a baseline mind state and that during meditation. When the value of this formula is high it indicates strong meditation.
2. The “eSense” (see below) values of Attention and Meditation did not differ significantly between baseline and performance segments of the session. The reason for this may be that Sally’s method of meditation produces gamma waves; her eyes are half opened and unfocused. eSense looks for alpha waves to indicate mediation; these are most easily produced with eyes closed. Sally suggested that all meditators do try to be attentive so the Attention ratio is not surprising (eSense looks for beta waves to measure attention). See [this](http://support.neurosky.com/kb/science/what-is-esense) for more information about eSense.
3. The total power output (across the first 8 waveforms) during meditation is about 40% of the baseline. This suggests that it is not the absolute strength of the gamma that is important, only its relative strength compared to the other waveform outputs.

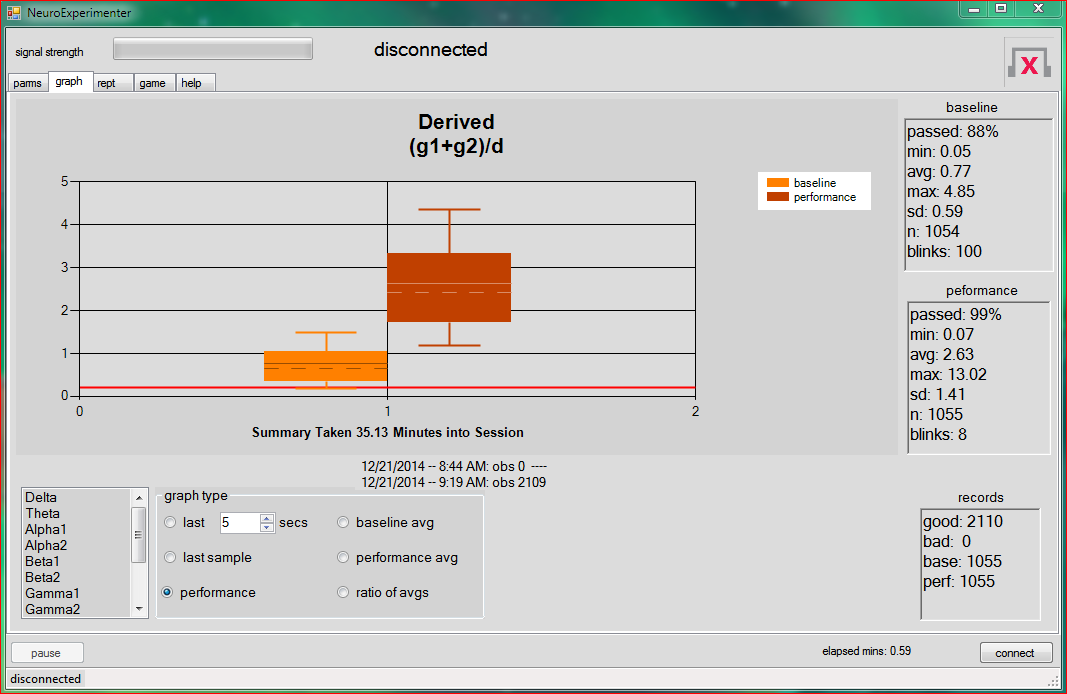
In order to determine the quality of our proposed formula we passed the log created during the session through NEx. Here is the setup for that run:



Now we have input a *derived formula* and selected the Derived waveform for performance feedback. The *replay log* function will use the original breakdown of the session (17 minutes for baseline), and ignore any values we might put in the timer boxes. After the log processing is finished we get this summary:



The Derived bar indicates our formula is good at discriminating the baseline from the rest of the session: meditation (=Derived) average output is about 3.4 times that of the baseline. More informative is the [boxplot](http://en.wikipedia.org/wiki/Box_plot) of the performance graph:

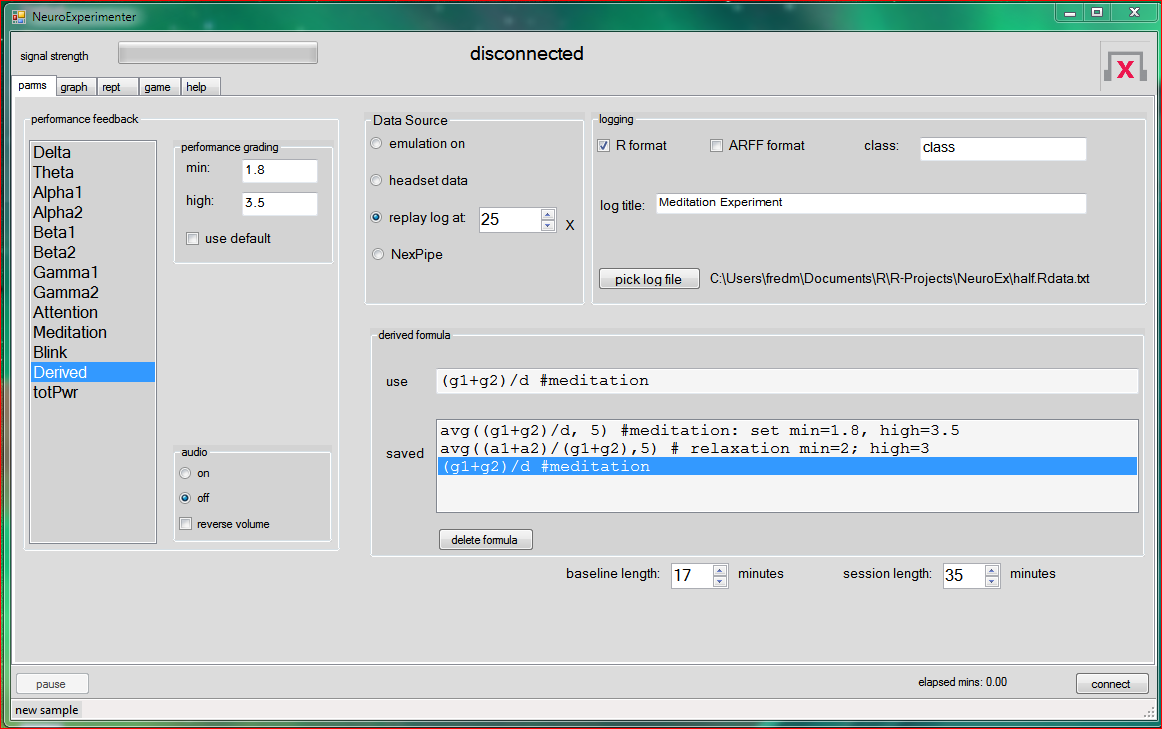


This indicates that 50% of the samples during the meditation had formula outputs between (roughly) 1.5-3.3; 90% were above 1.2; and 10% were above the top whisker at about 4.5.

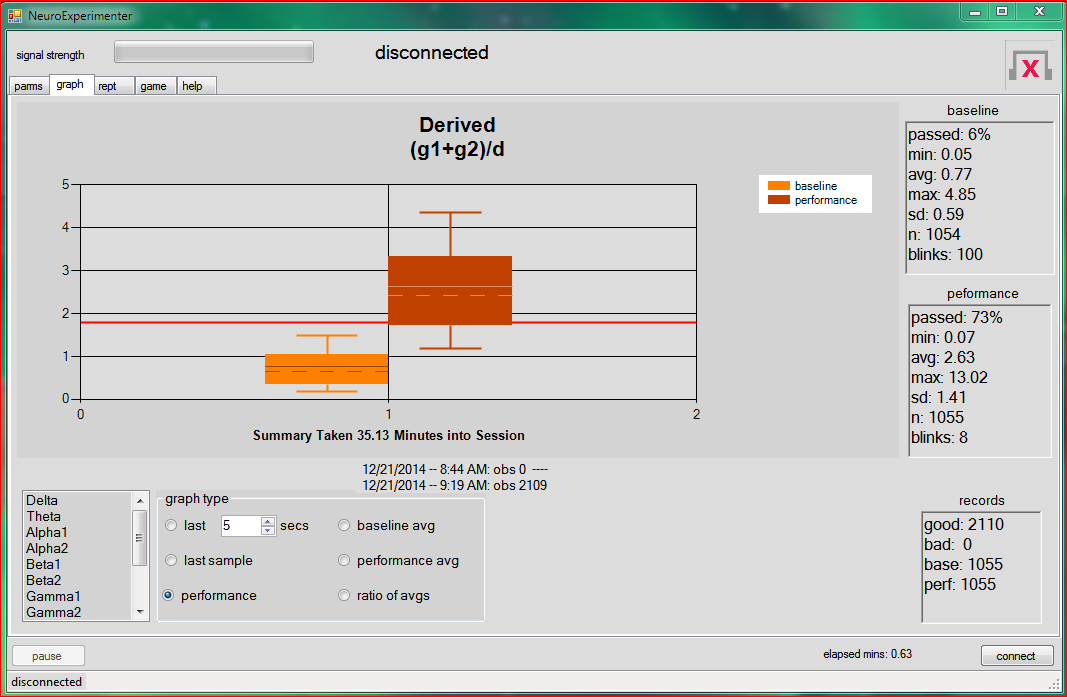
Less than 10% of the baseline data were above the top whisker at 1.5 (roughly). This suggests the grading parameters for training: the minimum score (i.e. Derived formula value for a sample) to signal meditation is about 1.8. A high score is anything above (say) 3.5. When a sample has output from the formula that is less than *min*, the sample “fails” (is non-meditation). If it is greater than or equal to the *min*, it “passes” (i.e. the subject is meditating).

Note that with the default value of *min* (.20), the above data shows that 99% of the performance data passed (that’s good), but 88% of the baseline data passed too (that’s not so good). The formula is fine, but the *min* value classifies too many of the baseline samples as meditation.

The next step is to see how well our formula along with new grading logic discriminates between the two sessions. The setup is:



The only changes we have made were to set the *grading* logic. When the replay is done, we go back to the performance graph for an assessment:



Now we see that only 6% of the baseline data would be classified as meditation whereas 73% of the performance data would be. Perhaps if we lowered the *min* we would pick up more meditation samples while not misclassifying too many more baseline ones. Maybe adding Beta2 to the numerator of the Derived formula would help. You can play with the Derived formula and the *grading* logic until you are satisfied that the classification is about as good as you can make it. Warning: if you make the formula too complicated you are in danger of [overfitting](http://en.wikipedia.org/wiki/Overfitting). Furthermore, trying for 100% is probably ill advised: it is unlikely that *all* of Sally’s mental state was in deep meditation.

I suggest (for this experiment) a Derived formula of avg((g1+g2)/d, 5). This smooths the formula output over the last 5 samples. When I put that in as the Derived formula it gives a baseline passed of 1%, but the performance passed increases to 82%. It will also smooth the audio output for a less erratic response -- helpful for training. See Appendix 1 for other functions you can apply in the Derived formula. Further analysis of this experiment is in Appendix 2.

The next step is to train someone in meditation. Retain the Derived formula, the *grading* logic, and the selection of the Derived waveform. Turn on the audio for feedback to the trainee. You should set *baseline length* to zero and reduce the time of the session to (say) 20 minutes. The trainee will put on the headset and start the session by trying to meditate. During the session he/she can receive audio and visual feedback (see below). The percent of samples that were classified “meditation” will be on the *performance* graph when the session ends, indicating how successful the trainee was.

## Aside:

The [effects](https://www.healthline.com/nutrition/12-benefits-of-meditation" \l "section11) of meditation on health are well documented. However, a [recent](https://www.nytimes.com/2019/03/14/health/alzheimers-memory.html) scientific

article claims that production of gamma waves can lesson the symptoms of Alzheimer’s.

The [article](https://www.nature.com/articles/d41586-018-02391-6) was published in *Nature* , a leading scientific journal. Perhaps a practice of

meditation that produces gamma waves can achieve the results too. See experiment 3

for an idea.

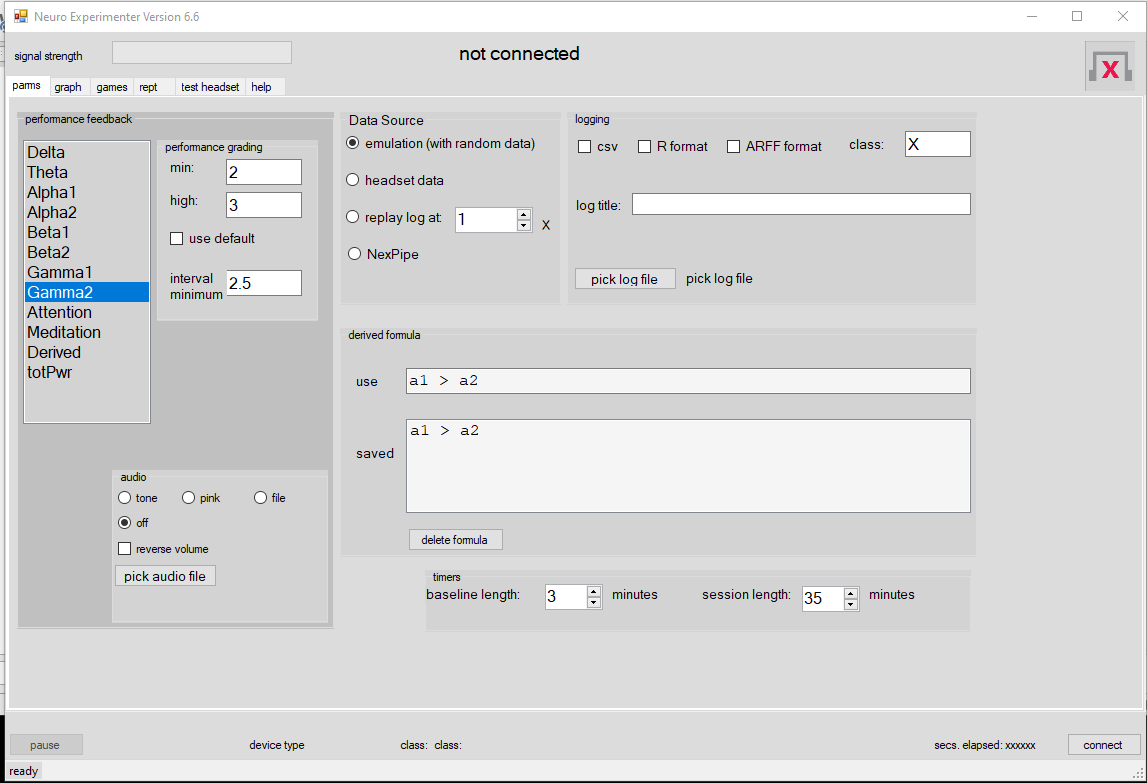
Sally asked me to include the following:

There is more to meditation than producing certain brainwaves. You must learn the internal and external cues that warn you of loss of focus. This headset might be useful to signal that your mind is wandering, but is no substitute for hard work and continual practice. Eventually, you will have to depend on yourself. There are many books, videos, and instructors available to guide you.

Other experiments like this one can be found in [Appendix 3](#1ksv4uv).

The next part of this Guide contains details about the NEx software.

# Parameters



## Data Source

You can run the application without the headset by clicking *emulation on*. Random data is generated by NEx. The random data is written to a log (if you so specify) This is the default*.*

Select *headset data* to acquire data from the headset. You can determine that the application is connected to the headset (even if it is off your head) as follows: the LED on the headset should be solid provided you have selected *headset data*, turned on the headset, and clicked on *connect*. You can pause the system while using the headset by clicking the *pause* button. This throws away subsequent samples sent by the headset and suspends logging and reporting. If NEx loses the headset signal while you are paused, you will be disconnected and the session ends. If you remain connected, click on the *pause* button (now labeled *resume*) to continue the session. No time is accumulated in baseline or performance sections until you resume.

You can replay a log in R format by selecting *replay log at*. This is a way to review any session for which you have created a log. You must specify an existing log via the *pick log file* button. The log can be played back at higher speeds by increasing the X multiplier. If you set it to 25 it will run as fast as it can (which is handy when you just want to see the results of tweaking the *Derived* formula and/or the *min* in the *grading* box). You can get an overview of the session by increasing the replay time and increasing the number of seconds (on the graph page). If you click on *pause* you can then reset the replay speed (on the *parms* page); the log continues at the new speed when you click the button again. This is useful if you wish to get to a certain point in the log quickly, then slow the replay so that you can examine the session slowly from that point.

You can change the selection of the performance wave type, the derived formula, and the grading logic before you start replaying the log. This allows you to perform different experiments against the same log. The timers are not used during the log replay: the baseline and session split are that of the original session.

Log files in other formats cannot be played in NEx. The output format is supplied so you can use

the log in other applications (the .csv log can be processed in R, Python, and spreadsheets like

Excel).

You can connect with devices besides MindWave via NexPipe. See Appendix 4 for details.

The input source (*emulation/headset/replay log*) cannot be switched once you have connected.

## Logging

You can save the sample data to up to 3 logs (except when replaying an existing log). Check the box(es) for the logs you wish. R format is suitable for input to [R](http://www.r-project.org/) (a free statistical system) and the NEx software. See Appendix 4 for the log format for R.

You can make a “csv” log with “comma separated variables”. This can be read into most spreadsheet programs as well as many other applications. Python has many capabilities for analyzing a csv log.

Note: only the R formatted file is suitable for using as a “replay” log. So, if you wish to replay the session be sure to create an “R format” log.

*Class box*

Log entries are tagged with the “class”. An ‘\*’ is appended for samples taken during the baseline. Furthermore, the class can be changed (to a single, printable character) by pressing a key on the keyboard while NEx is connected to the headset. All subsequent samples are tagged with this class until a different key is pressed. In this way you can mark samples with various external events (e.g. music starts to play, subject starts to answer a question, etc). You would then write a program/spreadsheet to analyze the log based on these events.

Remember that samples are taken about once per second. Hence the class will tag the next sample to be received by the headset, not the last sample taken.

## Log file

You can browse for an existing file (to overwrite or to read), or specify a new one (to be written to). Be careful: the last log will be overwritten when a new session is started, unless NEx is reading the log. To prevent overwriting, you must input a new file name before you click the *connect* button (alternatively, you could rename any log you wished to save).

Note: the filename you enter should not have extensions. NEx will remove all extensions and add new ones for each log selected. So, if you selected “/C:/myLogs/relax.Rdata.txt”, or input “relax”, NEx would give you a file “/C:/myLogs/relax.Rdata.txt” for the R-formatted log, and a file “/C:/myLogs/relax.csv” for the csv formatted log.

See Appendix 4 for the format of the R log file.

## Derived Formula

This allows you to create a new kind of waveform by combining other ones. See Appendix 1 for the format of the formula. The formula works on the *normalized* values of a data sample (wherein sample data is converted to values between 0 and 1). See Appendix 1 for the normalization process. The formula cannot be changed while data is being generated (i.e. while you are “connected”).

You can select an existing formula from the “saved” box. You can make a new formula by typing it in the “use” box. A formula is saved until you delete it with the “delete” button.

## Timers

The session can be divided into 2 parts: *baseline* and *performance*. The time for the whole session is specified by the session timer. Chimes are sounded at the end of the baseline and at the end of the session. When the session ends the headset is automatically disconnected. You can end the session at any time by clicking on the *connect/disconnect* button. The chimes are handy if you do not wish to look at the output graphs (or check your watch) during the session.

The baseline timer ends the first part of the session and starts the second part (the “performance" part). Set it to zero if you do not wish to divide the session into 2 parts. Summaries are kept and compared for the performance vs. the baseline data if you make use of the baseline timer.

The timers cannot be changed until you disconnect from the headset. When you pause the system no time is accumulated by the timers.

## Performance Feedback

Pick one of the wave types to measure for "performance". Your performance on that wave type will be measured and fed back to you. As explained in the example above, you will most often choose the *Derived* form. You need not select any of the waves but then feedback is limited. The selection cannot be changed during the session.

## Performance Grading

You can use the default, or choose upper and lower values to determine the grading logic. This is explained in Appendix 1 (below). The logic cannot be changed during the session. If it is in range, some of the graphs will show the *min* as a red line.

## Audio

It *tone* is selected, a steady tone is output for the last sample for the selected wave type. The

volume (or silence) and pitch reflect the intensity (power output) of the wave selected in the

feedback box. How the tone is generated is explained in Appendix 1. If you prefer, you can

select *file* instead of “tone”. The wave file picked (via the “pick audio button”) will be used

instead of a tone. In this case the pitch will not change, only the volume. Similarly, selecting *pink*

will output [pink noise](https://en.wikipedia.org/wiki/Pink_noise) .

# Output

Most of the output is based on normalized data (see below for headset operation).

## Audio output

Once the session starts, if you have selected a performance wave type and turned the audio on,

you will receive audio feedback. The tone’s frequency and volume will increase based on your

performance (see Appendix 1). If you check the “reverse volume” box the tone will get softer

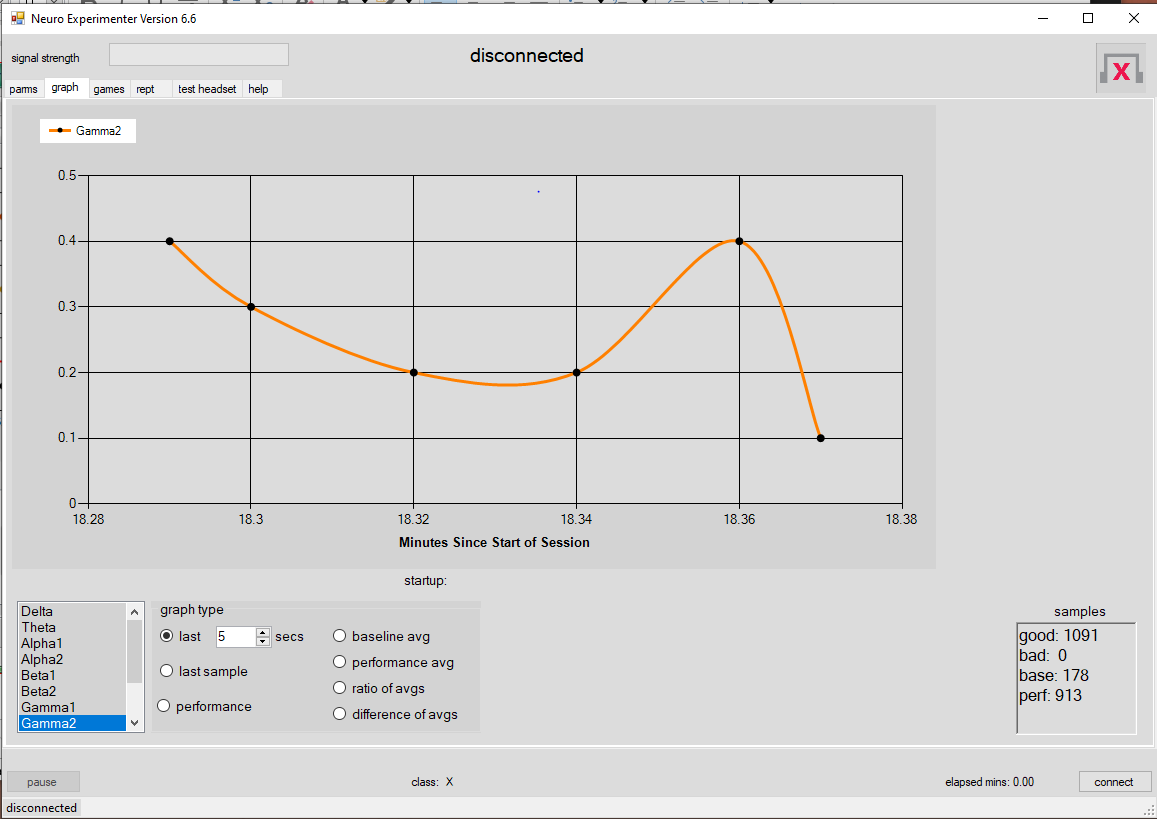
and lower as the signal strength increases. As mentioned above, you can pick a wave file or

pink noise instead of a tone.

You can turn the audio on or off during a session, provided you selected a type (i.e. not “off”)

before the session starts. However, the type cannot be changed until you disconnect.

## Graphical output



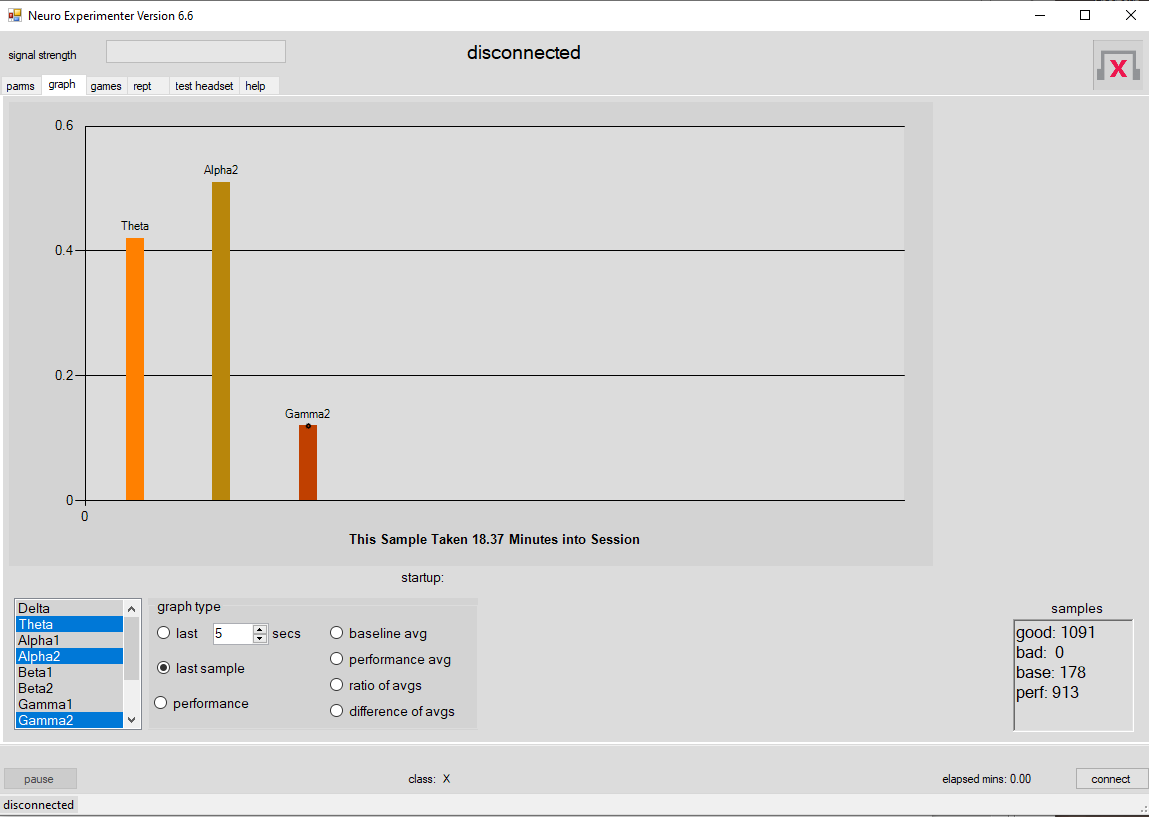
Select one or more of the wave data types in the graph box by clicking on it. Choose the graph type: either the last n seconds of output, just the last sample, a summary for the whole session, or performance data. Selections can be changed even while samples are being generated or a log replayed.

It might be unproductive to look at the graph while conducting an experiment. The audio feedback can be more suitable for training/feedback. When the experiment is over you can view the last graph produced, or use the “replay log” feature to replay the experiment and study the graph as it is being updated.

It is not too useful to select “totPwr” (which can have a magnitude in the millions) to graph with other waveforms since it is so large as to swamp the others (which are generally much, much, smaller).

This graph shows the last ‘n’ seconds (5 in the sample above) of the selected waveforms. All of the waveforms are normalized to be between 0 and 1, except for the Derived waveform (which uses the right scale) and “totPwr”. If the “performance” waveform is being graphed it is marked by black dots (Alpha2, above) and a red line for the *min.* (in the grading logic on the *parms* page) will be shown if it is in range. The unit of measure on the Y-axis is the normalized value (a

ratio. See Appendix 1).



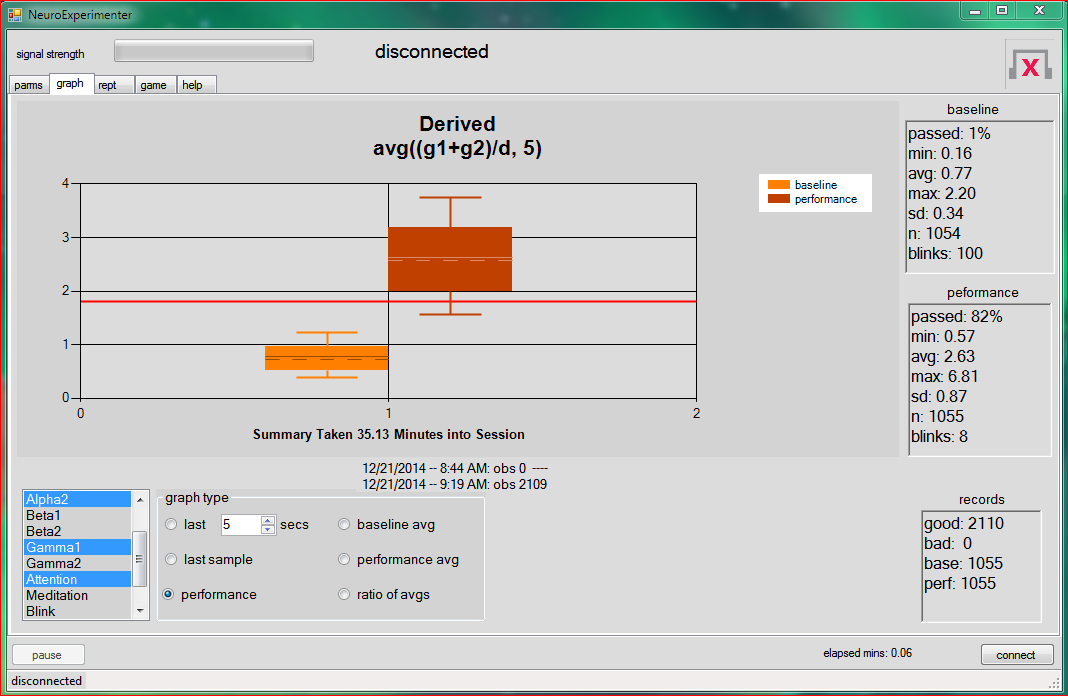
The *last sample* plot shows the last sample (for the waveforms selected). The *Derived* waveform uses the right scale. If the performance wavelength is plotted it is marked with a black dot (Alpha2 in this example). The unit of measure on the Y-axis is normalized ratio (see

Appendix 1).



This graph shows a summary. It is explained in the Example above. The bars represent ratios (if you have set the baseline time larger than zero). You can also see the baseline and performance averages separately. Although the chart is updated continuously, it is not very useful until the end of the session (as explained in the Example above).

An ‘\*’ indicates that the [difference](http://stattrek.com/hypothesis-test/difference-in-means.aspx) between the average (normalized) performance and average (normalized) baseline is statistically significant (to p=.01 level, 2-sided test). It was determined that the normalized power data and eSense data are not far from a normal distribution and so the test is valid when the number of samples is above 30. See the Appendix for details about the normalization logic.



The performance graph is not very useful until the end of the session, and only available if you have selected a performance wave type (on the parms page). It shows boxplots and summary data (including the percent of samples that passed the grading logic). If there is no baseline data, only one boxplot and one box of summary data will be shown. The pass/fail logic for classifying a sample is described in Appendix 1.

The dashed lines in the boxes are at the median of the data, the solid line is at the average. The

bottom and top of a box mark the 25th and 75th percentile; the lower and upper whiskers mark

the 10th and 90th percentile. The derived formula uses normalized values (see Appendix 1) so

the Y-axis shows a “pure number” (since it is derived from ratios, which are pure numbers).

The red line, when it is in range, shows the *min* selected for the *performance grading* (in the parms page), which determines pass/fail for a sample.

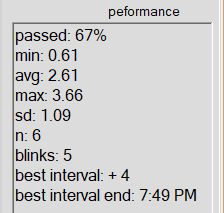
All the graphs show the *samples/records* box. The total samples: good, bad (as determined by the headset; bad samples are discarded by the NEx software), and the total good samples collected during the baseline and performance (post-baseline) parts of the session. When you are replaying a log, the “good” count is the number of records read.

Since samples can be combined in the logging process, this will not match the sample count in the original run. You can determine which part of the session you are in: either the *base* or *perf* sample counter

will be increasing.

You can determine which part of the session you are in: either the *base* or *perf* sample counter will be increasing.

Note: for version 4.7 we added two more lines in the performance box:



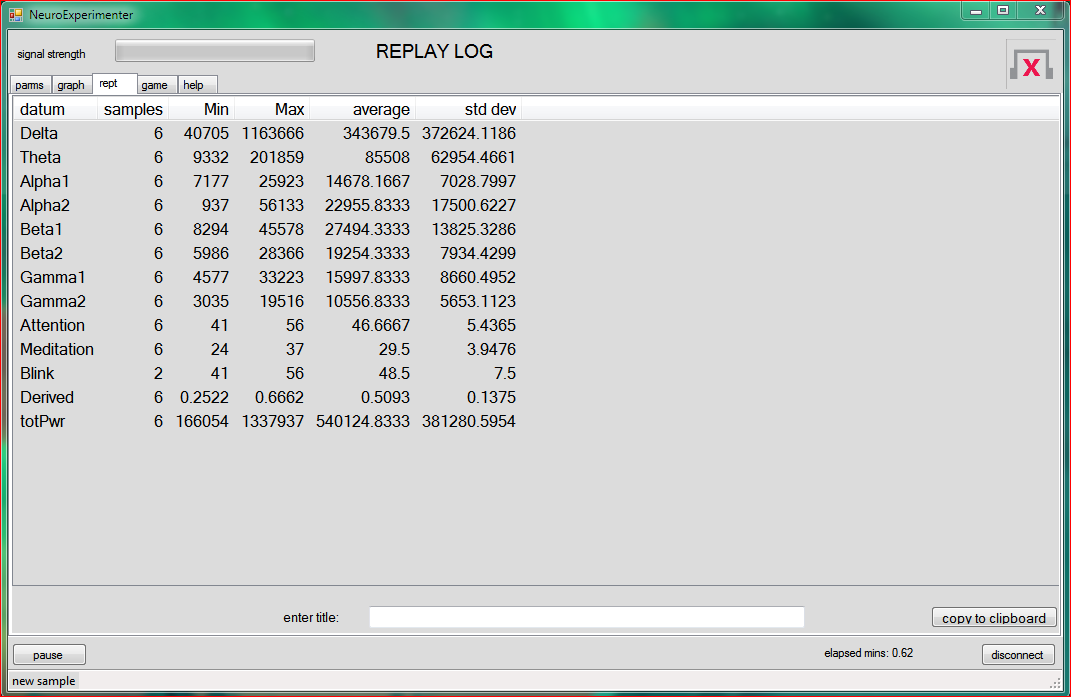
The “best interval” is the largest number of *contiguous* samples, all of which passed the grader’s

“interval minimum”. I.E. no sample in that interval was below the grader’s interval minimum. The

last line is the time at which that interval ended. The “interval minimum” is entirely independent

of the grader’s “min” and “high” parameters.

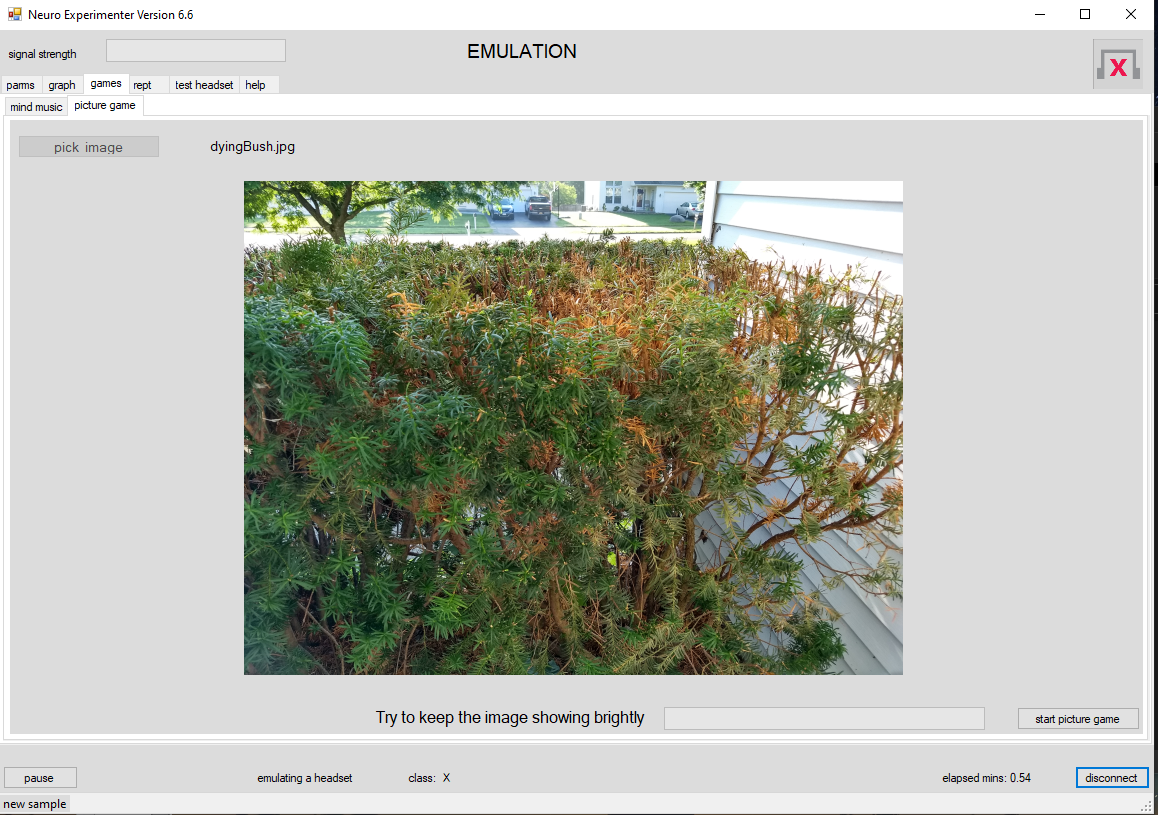
## Report



You can click on the *rept* tab to see a report. If you wish to save the report you can enter a title and click on *copy to clipboard*. Then, open another application (like Notepad) and copy to it (via CTL-V). All of the waveforms are shown. The data is not normalized; it is the power data returned by the headset. The data is updated continuously, but is frozen while you are on the *rept* page.

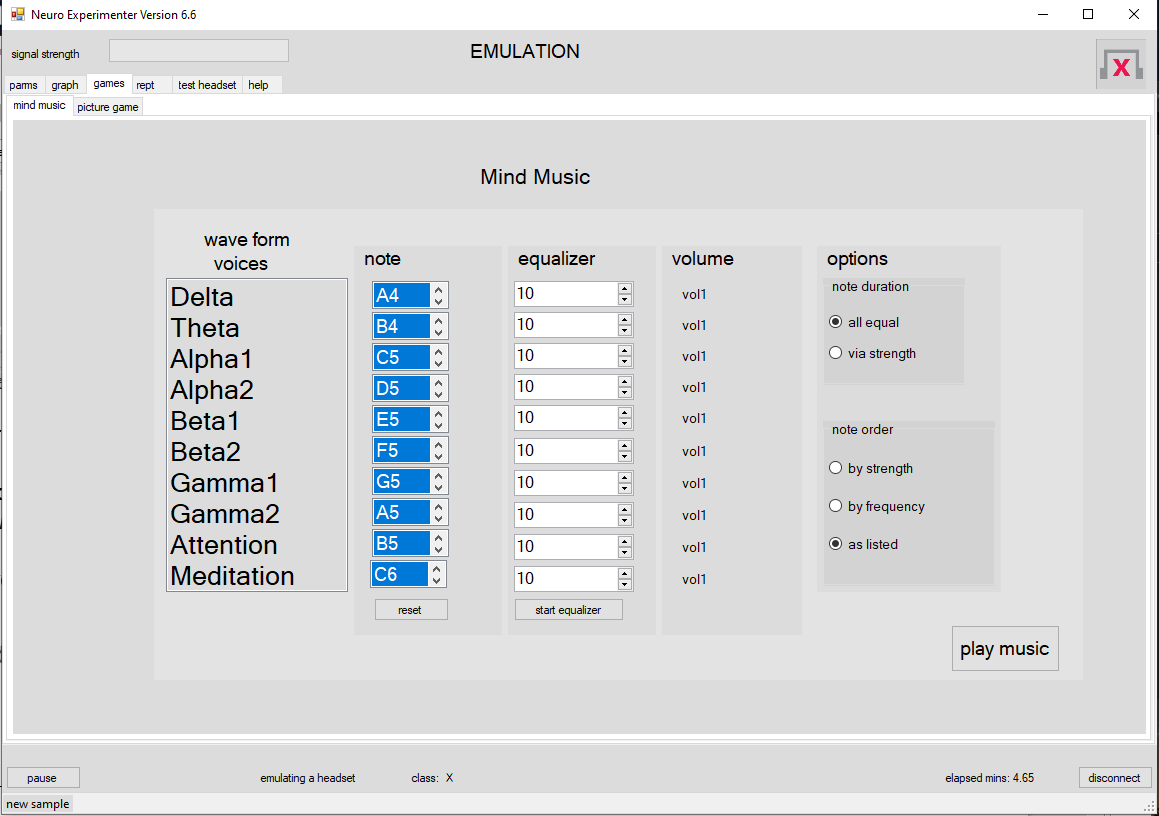
# Games

## Picture game



Before you click on the *game* tab you should have submitted a formula in the *derived formula* box. To start the game, click on the *start picture game* button. If you have not done so alreadly, NeuroEx will ask you to pick an image file from your computer. You can change this with the *pick image* button (when you are not connected). Your goal is to make the image bright and clear by trying to increase the strength of the *Derived* waveform (the strength is shown by the progress bar). Start the game by clicking on the *connect* button; stop by disconnecting.

## mind music



Start by clicking on the *wave form voices* you want monitored. You can select the *note* for each of the wave forms. To see how the music will play, click *start equalizer*. The notes you have selected will play so that you can adjust the volume in the *equalizer* boxes.

When you are satisfied, select the *options* in order to control the duration of each note, and the note order: by *strength* means strongest waves are played first.

Turn off the equalizer, click *play music*, and connect to the headset. Now “music” will play based on the waves output from the headset (and their notes) that you have selected.

Notes are played one after the other (not simulantiously). Hence the time to play all the notes may extend over the next signal from the headset (which are created every half-second). Hence the music will be a delayed representation of your mind waves.

# Running the application

Once you have chosen emulation, headset, or replay log click on the *connect* button. Its name will change to *disconnect*. You will be shown the graph page. To stop the application click on the *disconnect* button. Each click of the connect button determines a “session”. The log, graphs, and report are all started over when a new session begins.

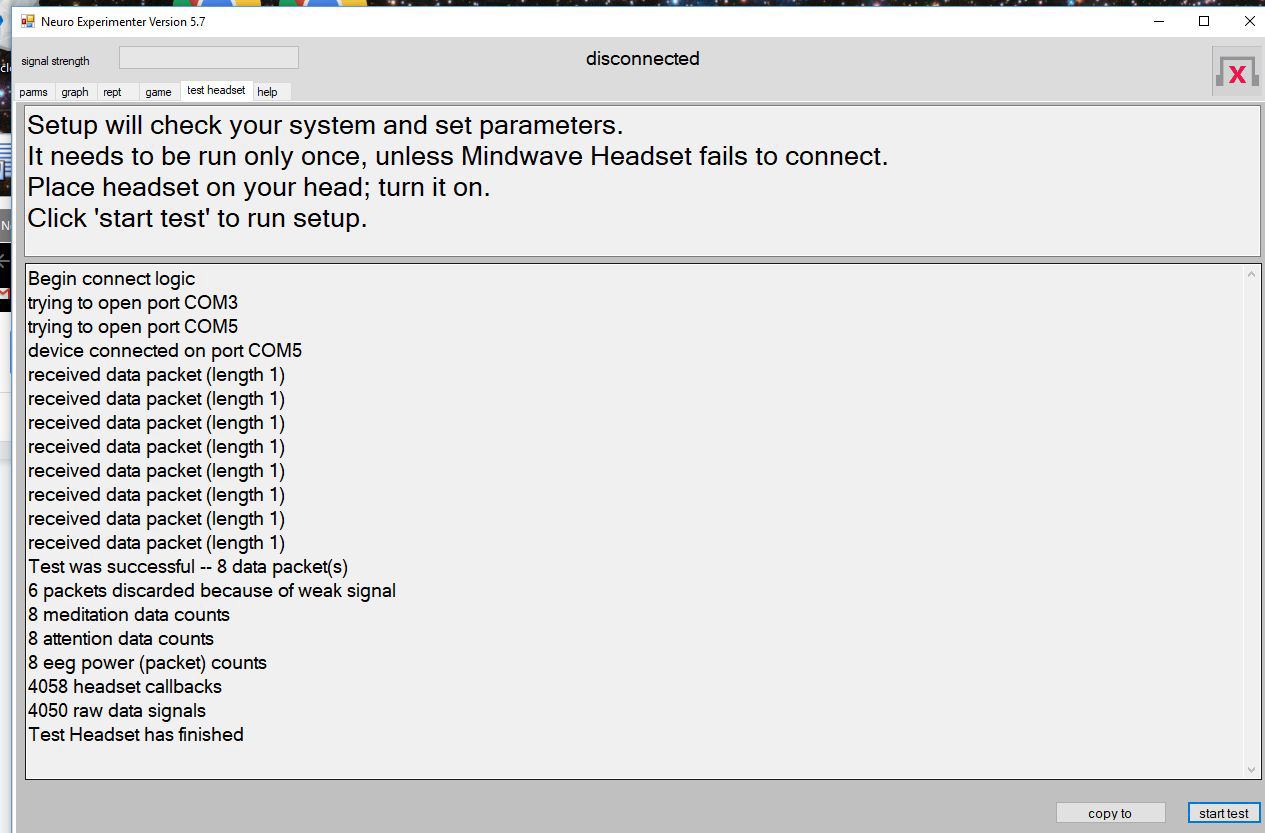
## Test (test headset tab)

This is provided to test the communications between the headset and NEx. If you have

problems this can help diagnose them.

Place the headset on your head, turn it on, and click on the “start test” button. A successful test

will produce output that looks somewhat like this:



You can click the “copy to” button to copy the output to the clipboard. You can then paste into

email or a text file. Failure to connect may have the following causes:

1. Headset not seated properly on the head, or not turned on.

2. Weak batteries in the headset.

3. Improper bluetooth setup. See Appendix 5 .

# Headset Data

The following information was derived from the Neurosky documentation and discussion with their support group.

The headset generates “raw” data about 512 times a second. “Power” data are generated about once per second. All 8 power data (one for each wave type) are generated at the same time. Meditation and Attention data are also generated about once per second, but not necessarily at the same time, or at the same time as the power data. Attention and Mediation data (“eSense” data) are calculated by the headset in an unknown manner (perhaps from the power data and/or the raw data). Raw data is not used by NEx.

In EEG parlance, the “power spectrum” is calculated by breaking the signal into its component frequencies (Fourier analysis), and the square of the amplitude (as a function of each frequency -- alpha, theta, etc) is then the “power spectrum”. See <http://en.wikipedia.org/wiki/Quantitative_electroencephalography>

According to the NeuroSky documentation:

[The power data] values have no units and therefore are only meaningful when compared to each other and to themselves, for considering relative quantity and temporal fluctuations.

All data from the headset are plotted as they are received. However, data for the log are buffered and accumulated. As data is delivered to NEx the values are place in a buffer. Data is not written to the log file until new data arrives which would overwrite some of the buffered data. This allows us to combine power data and calculated data (Attention, Meditation, Blink) into a single record.

The timestamp for this data is that of the last datum that arrived. Because power data arrives at least every second, and so causes the buffer to be written at least once per second, the timestamp will be accurate to the second. Be aware, however, that the data in the buffer were not generated at exactly the same time by the headset (just all within the same second).

Typically, a set of power data and Attention and Meditation will be combined, no blink data will have occurred and so Blink will be set to NA. When a blink value arrives, if the buffer has no blink data, it will be combined with other data in the buffer. Sometimes a log record will be created with only blink data (all other data set to NA). This can happen if we have (say) three blinks within the same second. This buffering strategy can result in some log records having the same timestamp (which, again, is only accurate to the second).

*Normalization*

In order to compare waveforms across samples, and for the Derived formula to make sense, most data are normalized to be between 0.0-1.0 (see the Appendix). Attention and Meditation are delivered by the headset as values between 0-100. These are divided by 100 to normalize them.

Thus all data, except for Derive and “totPwr” will have values between 0 and 1.

The Derived formula operates on the normalized data and is not itself normalized. To show it on the same graph as the power data, another Y-axis (on the right) is sometimes used.

The wave type “totPwr” is not normalized (see the Appendix).

For the report and for the logs, the samples are not normalized. The actual values from the headset are reported.

Here are the EEG band frequencies produced by the headset (these are standard across EEG literature):

Delta: 1-3Hz

Theta: 4-7Hz

Alpha1: 8-9Hz

Alpha2: 10-12Hz

Beta1: 13-17Hz

Beta2: 18-30Hz

Gamma1: 31-40Hz

Gamma2: 41-50Hz

Further information about the headset is found [here.](http://support.neurosky.com/kb/science)

# Conclusion

In a perfect world neurofeedback training could achieve the following:

1. You train yourself to fall asleep instantly by learning to achieve the “relaxed” state.
2. You can maintain focus and your mind does not wander because you have trained yourself in the “concentration” state.
3. You don’t become angry because you have measured your brainwaves when you are angry and when you are calm. You can achieve the “calm” state when you feel anger approaching.
4. You cure your problems of addiction, ADHD, anxiety, depression, and migraine all through neurofeedback.

The above is hyperbolic. How much is really possible? Here are only *some* of the problems:

1. The significance of particular brain waves depends on where in the brain they are produced. Medical EEGs can determine this, but the headset has but one contact and so does not give us locations.
2. Even when the headset does feed us pertinent information, discriminating criteria may be buried so deep in the data that all of our analysis with **R (**or with machine learning software) cannot discover them.
3. Even if we discover a formula and the headset gives us the appropriate signals, it might be impossible for a person to generate the waves at will. There are limits to our control of our physiology.

The history of [neurofeedback](http://en.wikipedia.org/wiki/Neurofeedback) extends from 1924 to the present day. [Biofeedback](http://en.wikipedia.org/wiki/Biofeedback) goes back to 1875. Strong claims have been made. It seems reasonable to expect that EEG data can translate into “mind states” with some accuracy. [Here](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.683&rep=rep1&type=pdf) is a good example of an hypothesis verified by experiment.

The Mindwave headset, this software, and **R** will let you experiment and test hypotheses. It offers a limited capability for self-training. Do not expect unrealistic results. Keep an open mind <:-)> but strictly adhere to the scientific method.

# Bugs and Suggestions

Please contact the author, Fred Mellender, at [fredm73@gmail.com](mailto:fredm73@gmail.com) to report bugs or suggestions. Please see Appendix 5 for known problems and how to report bugs.

This software is free, but no guarantee as to its suitability or correctness is given. You can make copies of the program and distribute it freely.

This software and documentation are copyright 2021 by Fred Mellender and are *not* to be sold.

**Appendix 1**

# **Normalized Data**

According to the Neurosky staff, data between sessions are not easily compared. This is because the contact between the headset and the user will differ as environmental conditions change. Furthermore, the total power output (i.e. sum of the power of the 8 standard waveforms) varies from sample to sample.

I confirmed this variability. I summed the power values for each sample in two different sessions, lasting about 30 minutes each. The data for these “sums of power for each sample”, or “total sample power” (calculated by **R** from the NEx logs:

sums <- log1$Delta+log1$Theta+log1$Alpha1+log1$Alpha2+log1$Beta1+

log1$Beta2+log1$Gamma1+log1$Gamma2)

were:

Session 1:

> summary(sums)

Min. 1st Qu. Median Mean 3rd Qu. Max.

27660 59790 93680 252100 219000 4287000

> sd(sums)

[1] 436093.6

Session 2:

> summary(sums)

Min. 1st Qu. Median Mean 3rd Qu. Max.

20660 60860 81840 164800 129600 3926000

> sd(sums)

[1] 304046.3

You can see that within a session and across sessions the total power data can vary quite widely. In any case, it is expected that the following normalization process will allow us to compare samples within and across sessions. It is the technique recommended by one of the NeuroSky staff. (Normalization is surprisingly tricky, as a search on Google -- “normalizing eeg data”-- will confirm. Just for starters, don’t even think about comparing unnormalized data between individuals).

The NEx program normalizes the power data by first summing all the power data (i.e. meditation, attention, and blink are excluded in the sum), to get “total power” for the sample. Then each power datum in the sample is divided by this total power. Then the square root is taken of the result. Each of the normalized datum in a sample is thus the square root of the proportion of that datum to the total power of the sample. It is between 0.0-1.0. Taking the square root has the effect of making the power data (approximately) normally distributed. This allows us to do the test for significance of the difference of baseline and performance data. (Significance at the p=.01 level, 2-tailed test, is indicated by an ‘\*’ on the summary chart).

The total power (unnormalized) can be viewed by selecting the “totPwr” wave type. However, it is not useful to graph this with other data since they will be swamped (average “totPwr” is in the neighborhood of 500,000).

The NEx graphs show a normalized value for the 8 power values.

The Attention and Meditation data are simply divided by 100 since these values are already scaled to be between 0 and 100 by the headset. These data are (approximately) normally distributed. BlinkStrength is divided by 255. The Derived data is not normalized, but the right hand scale on the graph is used. The tone to play is calculated as for the other waveforms (see below).

So, the question remains how we grade for (say) a “high Alpha” and then sound the appropriate tone? This is explained in the next section.

# **Performance Grading and Tone Generation**

The purpose of the tone is to give audio feedback while a training session is in progress. The value for generating the tone is taken from the normalized data (where each waveform except for the Derived one) is scaled as explained above. No audio feedback or performance grading is done unless a wave type has been selected (in the *performance feedback* box).

If the user selects the *default* grading, *min* is set to the average of the samples of the wave type selected; *high* is set to the .90 \* maximum of the samples. Because the average and maximum are recalculated as new samples arrive, they can change. Thus audio feedback will change during the session when the *default* logic is selected. Things should settle down eventually. If you don’t use the *default* logic the *min* and *high* do not change: they are taken from the user input.

If it is in range, the *min* will be shown on some charts as a red line.

Once we have a *min* and *high* (calculated or input directly by the user), we can calculate a frequency and a volume for the audio. These values depend on how far above the *min* the performance waveform’s strength is. If is above the *high* value, volume and frequency are set to the maximum. If it is below the *min* value, the audio is turned off. Both frequency and volume decrease as the strength drops off from the *high*.

The strategy for the audio output is reversed if the *reverse volume* box is checked. Audio is turned off if the performance is above the *high* value, and frequency and volume are increased as the performance waveform’s strength decreases below the high value (i.e. silence is good).

You can select the Derived waveform to generate the tone. However, for boolean values (e.g. a1>d) the tone will be either on (at maximum volume and frequency) or off, because the value is either 0 or 1. If you want to scale this example to generate different tones, you could try (a1>d)\*a1, max(0, (a1-d)), or various alternatives.

Note that the logic implies that the higher the strength of the performance waveform, the “better”, in the sense of a louder, higher frequency tone. All samples below the minimum are scored as “not passed”; all samples above the minimum “pass”. If this is not what you want (e.g. most baseline samples are *above* most performance samples for your Derived formula), just negate the formula involved (via the neg function); you could take the reciprocal instead (i.e. 1/formula).

# **Derived Formula**

This allows you to specify a combination of wave data from the (normalized) sample to generate another waveform (called Derived) for graphing and audio output. The formula must be input according to the following “[BNF](http://en.wikipedia.org/wiki/Backus–Naur_Form)” rules:

1. Waveforms to combine are specified by the following variables (VAR):

"d", //"Delta",

"t", //"Theta",

"a1", //"Alpha1",

"a2", //"Alpha2",

"b1", //"Beta1",

"b2", //"Beta2",

"g1", //"Gamma1",

"g2", //"Gamma2",

"att", //"Attention",

"med", //"Meditation"

Because the VAR data are normalized before they are used in the formula their values are between 0-1.

2. Operations (OPER) between waveforms (VARs) and other expressions are specified thus:

"+", //add

"-", //subtract

"\*", //multiply

"/", //divide

"<", //less than

">", //greater than

"|", //or

"&", //and

“=”, //equal

3. Additional “terminals” (constants) are

LP- ‘(‘

RP- ‘)’

COMMA - ’,’

NUMBER- (e.g. 4.56)

FUNC1 - abs, sqrt, log, neg (negate), and ‘!’ (not)

(all FUNC1 take one parameter)

FUNC2 - avg, min, and max (takes 2 parameters)

FUNC3 - if (if-then-else, takes 3 parameter)

4. There are only 9 rules for forming a Derived formula (expression). Here they are, followed by the rule number:

"expression : listUnit", //0

"listUnit: unit", //1

"listUnit: unit OPER listUnit", //2 //right association

"unit : NUMBER", //3

"unit : VAR", //4

"unit : LP expression RP", //5

"unit : FUNC1 LP expression RP", //6

"unit : FUNC2 LP expression COMMA expression RP", //7

“unit : FUNC3 LP expression COMMA expression COMMA expression RP", //8

Because of the recursive nature of the rules, an expression can be of any length and nesting level. The Derived formula you enter must be an expression as defined by the BNF.

You can append the formula with a comment: just precede it with a ‘#’, as in

a1+a2 #this is the sum of Alpha1 and Alpha2

All expressions (thus all formulae you can enter) will evaluate to a number (a double precision floating point number). When an expression generates TRUE or FALSE (e.g. a1>a2) it is translated to 1.0 or 0.0 respectively. This allows boolean expressions to be used in arithmetic ones, as in (d>t)\*a1, which translates to either 0.0 or a1 depending on whether d is greater than t.

The four single parameter functions (FUNC1) are abs (the absolute value, as in abs(d-t)), sqrt (square root), log (natural logarithm), neg (negate) and finally ‘!’ (not; e.g. !(d=t) means d is not equal to t). Note: the ‘-’ is reserved for the subtraction operation. To negate an expression (or just a number), use the neg function (e.g. neg(3)).

There are three FUNC2 available. The first one is avg. This is a binary function; the first parameter is any expression, the second is the number of samples (‘n’) to average over. For each sample from the headset, the expression is evaluated and the last ‘n’ values saved. These ‘n’ data points are averaged and the resulting number output as the value of the avg function. Here is an example:

avg(d>t,5)

For each sample, delta is compared to theta. When the former is larger than the latter, the expression evaluates to 1; otherwise it is 0. The last 5 calculations are kept and the output will be the average of those five (and hence a number between 0 and 1 in this case). The expression (first parameter of avg) can be of any complexity. The main use of the avg function is to smooth out the audio response as the waveforms involved in the expression fluctuate. This is useful in training sessions.

The other two binary functions (FUNC2) you can use are min and max, which return the minimum or maximum of the two parameters. E.G. min(a1,a2)

The one three-parameter function is if. The first parameter is tested as a boolean. If it is true (not zero), the if expression equates to the second parameter, else it evaluates to the third one. Thus

if((a1=b1), 3, a1\*b1)

translates to 3 if a1 is equal to b1, and to a1\*b1 otherwise.

To calculate a Derived value, each sample (containing all the 10 waveforms) is evaluated by the BNF rules and a number is returned. Note that the sample is processed after normalization (so that each variable (VAR) value is between 0.0 and 1.0). The resulting expression can evaluate to any number, even negative. Here are some examples of formulae and their meaning:

1. d > t If delta is larger than theta, return 1; otherwise this formula evaluates to 0.

2. (d > t) & (d > .5) Return 1 if both delta is larger than theta, and delta is larger than .5. Otherwise return 0.

3. (d>t) | ((d \* t) > .5) Return 1 if either delta is larger than theta, or delta times theta is larger than .5 (or both). Otherwise, return 0.

4. d The Derived waveform is the same as the delta waveform.

5. avg(d,5) Return the average of delta over the last 5 samples.

6. avg(d,5) > .5 Return 1 if the average of the last 5 delta samples is larger than .5. Else return 0. If you are trying to generate strong delta waves, more or less consistently, this would give you reasonable audio to train with. You could generate a continuous tone which depends on the strength of d (rather than just on or off) via (d>.5)\*d. The tone will be off if d is not greater that .5, but will depend on the strength of d if it is greater than 5. You could average this too.

7. ((a1+a2)/2) > (b1\*.5) Return 1 if the average of Alpha1 and Alpha2 is larger than half of Beta1; else return 0.

**Please note the following:**

The formula may be nested to any depth, but *parentheses are required* to order the operations. For example,

a1 \* b1 + 3

will be parsed as

a1 \* (b1 + 3)

because the parser uses “right association”. Parentheses can get you what you intend:

(a1 \* b1) + 3

It is probably best to use parentheses around each “unit” so that you get what you want.

Parse errors are indicated by a carrot (‘^’), as in

bad formula: parse error at pos 16: avg((g1+g2)/d, 5^, 6)

The carrot (‘^’) is the spot where the parse failed (in this case ahead of the ‘,’). This error resulted because avg takes only 2 parameters.

**A good formula**

How do you know if your Derived formula is a good one? Look at the boxplot on the *performance* graph. You are trying for as little overlap as possible between the ranges of the baseline and performance data, and hence between the boxes’ whiskers. If the top whisker of the baseline were the same as the lower whisker of the performance, you would have 90% of the baseline samples lower than 90% of the performance ones. That is as much as the boxplot can tell you when you have a good formula. Depend on the *passed* percentages in the baseline and performance data boxes to measure how good the formula (along with the *min*) actually is.

If you are not working with a baseline, the best you can do is make sure that your formula covers a large percentage of the observations in a log which you feel represents the mind state you wish to achieve.

As you progress you may find you need to adjust the formula and/or the *min* to make the sessions more challenging. The mind state you are going for might be pretty far from the one you started with (because you have improved with practice).

**Appendix 2**

Below is “Knitr” output from [R](http://www.r-project.org/), a free statistical package. [Coursera](https://www.coursera.org/) offers free courses on learning statistics with **R**. [YouTube](https://www.youtube.com/watch?v=6jT6Rit_5EQ) has many tutorials too.

Knitr renders an “Rmd” file by interspersing English text, R code, and the results of executing the R code. See [this](http://rmarkdown.rstudio.com/) for more information.

I duplicated some of the functions we did in the *Illustrative Example* that starts this Users’ Guide. The rendered file appears below. The R-code is in the Consolas font, the output from that R-code is preceded by ‘##’. My comments are in the current font.

I show how you can read in the log from NEx, eliminate unwanted variables, and perform the same normalization that is done by NEx. A “class” variable is added to each row to indicate that it was either a baseline or performance datum. I added a column, funct, that duplicated the results of our Derived function (g1+g2)/d, and verified that we get the same results using R that we got via NEx.

Note, however, that the log does contain the Derived data as well as the *totPwr* so you need not recalculate them in your own scripts. Furthermore, the log contains the *class* input appended with an ‘\*’ for baseline data, so you need not make separate data values for the class either.

There are various machine learning functions in R. One of the most effective is [“random forest”](http://en.wikipedia.org/wiki/Random_forest). The [confusion matrix](http://en.wikipedia.org/wiki/Confusion_matrix) shows that on our “testX” data, it classified 88% of the 271 base data correctly (NEx was 96% accurate), and 87% of the 265 meditation samples correctly (NEx was only 70% accurate, but we can boost this by taking the avg -- to get 82% accuracy-- and/or by playing with the formula and the grading logic).

All in all, our “Derived formula” classifier in NEx is pretty good. What’s more (unlike Random Forest) it is easy to construct and easy to understand how it processes an observation.

Learning R is well worth the trouble. You can use it to process the log data from NEx in many ways. Use the above script to get you started. It (and the experiment’s data) are included in the download of NEx. You will need to install [RStudio](http://www.rstudio.com/products/RStudio/). Then, you can click on NeuroEx.Rproj and see/edit the script.

**Meditation Experiment with NeuroExperimenter**

**Fred Mellender**

**Saturday, December 27, 2014**

**require("knitr")**

**## Loading required package: knitr**

***#opts\_chunk$set(eval=FALSE) #evaluate nothing***

### Read and clean the data

**half <- read.table("./half.Rdata.txt", header=T, na.strings="NA") *#first half: baseline, then meditation***

**clean <- function(df) { *#eliminate useless columns***

**df$class <- NULL**

**df$Blink <- NULL**

**df$time <- NULL**

**df <- df[complete.cases(df),] *#retain only those rows that have NO NA's in them***

**df**

**}**

**half <- clean(half)**

### Normalize the data

Divide the raw data from the log by the normalizing "totPower", just as we do in NEx:

**totPower <- function(df) { df$Delta+df$Theta+df$Alpha1+df$Alpha2+df$Beta1+df$Beta2+df$Gamma1+df$Gamma2}**

**normal <- function(df) {**

**foo <- df**

**sums <- totPower(df)**

**foo <- sqrt(df/sums) *#normalization of the power data (and Meditation/Attention, incorrectly)***

**foo$Meditation <- df$Meditation/100.0 *#do the Meditation normalization correctly.***

**foo$Attention <- df$Attention/100.0**

**data.frame(df=foo, sums=sums) *# return a new dataframe that includes the totPwr***

**}**

**normHalf <- normal(half)**

**head(normHalf, 2) *#show the first two rows of the normalized data***

**## df.Delta df.Theta df.Alpha1 df.Alpha2 df.Beta1 df.Beta2 df.Gamma1**

**## 1 0.8575289 0.4506945 0.1546900 0.06671388 0.08456472 0.1021114 0.09175004**

**## 2 0.8701965 0.3826817 0.1293046 0.19298949 0.10055612 0.1093220 0.11702737**

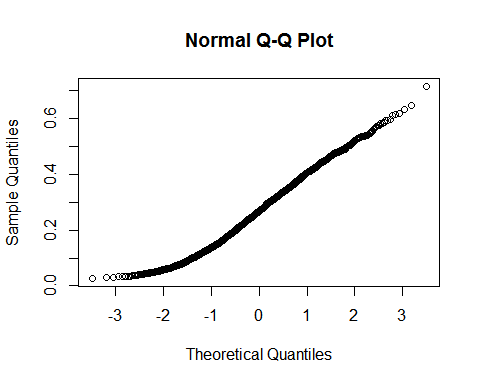
**## df.Gamma2 df.Attention df.Meditation sums**

**## 1 0.08451543 0 0 119980**

**## 2 0.08117834 0 0 1317011**

To do the test for significance we need to ensure that the normalized data it not too far from a normal distribution. I did tests on all the data; here it is for Gamma1.

**qqnorm(normHalf$df.Gamma1)**



The near-straight line suggests normality.

Add a class variable to distinguish baseline from mediation observations. The meditation observations begin at row 1055

**normHalf$class <- "base"**

**lastHalf <- 1055:nrow(normHalf)**

**normHalf[lastHalf,]$class <- "med"**

**normHalf$class <- as.factor(normHalf$class)**

Setup Derived formula and apply it to get another column to discriminate on. Split the dataFrame normHalf into two new ones (the baseline and meditation data).

**normHalf$funct <- (normHalf$df.Gamma1 + normHalf$df.Gamma2)/normHalf$df.Delta**

**base <- normHalf[1:1054,]**

**med <- normHalf[lastHalf,]**

### Explore the data.

We redo the NEx calculations in R to verify R and NEx match in their logic.

**head(base, 2) *# just show the first two rows of the data frame for the base data***

**## df.Delta df.Theta df.Alpha1 df.Alpha2 df.Beta1 df.Beta2 df.Gamma1**

**## 1 0.8575289 0.4506945 0.1546900 0.06671388 0.08456472 0.1021114 0.09175004**

**## 2 0.8701965 0.3826817 0.1293046 0.19298949 0.10055612 0.1093220 0.11702737**

**## df.Gamma2 df.Attention df.Meditation sums class funct**

**## 1 0.08451543 0 0 119980 base 0.2055505**

**## 2 0.08117834 0 0 1317011 base 0.2277712**

**baseAvg <- apply(base[,1:10],2,mean) *#get the averages of the first 10 cols of base***

**medAvg <- apply(med[,1:10],2,mean) *#... and of meditation***

**medAvg/baseAvg *# get the ratios, just as in NEx Summary graph page***

**## df.Delta df.Theta df.Alpha1 df.Alpha2 df.Beta1**

**## 0.5355591 0.8538690 1.1132707 1.1510909 1.1831242**

**## df.Beta2 df.Gamma1 df.Gamma2 df.Attention df.Meditation**

**## 1.6481569 1.8664350 1.8790000 1.0895014 0.8461755**

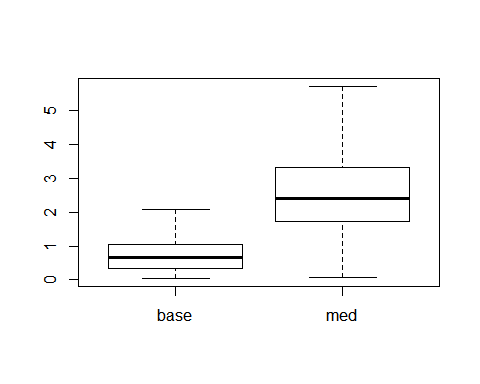
**mean(base$funct>=1.8) *# get the % passed of the baseline data***

**## [1] 0.06072106**

**mean(med$funct>=1.8) *# get the % passed of the meditation data***

**## [1] 0.7267552**

**boxplot(funct ~ class, outline=F, data = normHalf) *#get the boxblot like that on performance page***



This verifies that we get the same results with the formula that we got via NEx.

Now eliminate the sums, funct, Attention, Meditation columns from normHalf so we can do machine learning on what's left.

**normHalf$sums <- NULL**

**normHalf$functs <- NULL**

**normHalf$df.Attention <- NULL**

**normHalf$df.Meditation <- NULL**

### Divide into training and test sets.

We will train on the trainX set and evaluate on the data we held out from training (testX).

**library(caret)**

**## Loading required package: lattice**

**## Loading required package: ggplot2**

**inTrain <- createDataPartition(y=normHalf$class, p=0.75, list=FALSE)**

**trainX <- normHalf[inTrain,]**

**testX <- normHalf[-inTrain,]**

### Train a random forest model.

We train on trainX part we extracted from normHalf.

**model <- train(class ~ ., method="rf", data=trainX) *#random forest, which is also the default method.***

**## Loading required package: randomForest**

**## randomForest 4.6-10**

**## Type rfNews() to see new features/changes/bug fixes.**

**print(model)**

**## Random Forest**

**##**

**## 1582 samples**

**## 9 predictor**

**## 2 classes: 'base', 'med'**

**##**

**## No pre-processing**

**## Resampling: Bootstrapped (25 reps)**

**##**

**## Summary of sample sizes: 1582, 1582, 1582, 1582, 1582, 1582, ...**

**##**

**## Resampling results across tuning parameters:**

**##**

**## mtry Accuracy Kappa Accuracy SD Kappa SD**

**## 2 0.8638493 0.7275377 0.01022461 0.02035124**

**## 5 0.8578909 0.7155978 0.01054382 0.02105292**

**## 9 0.8510728 0.7019672 0.01151499 0.02296292**

**##**

**## Accuracy was used to select the optimal model using the largest value.**

**## The final value used for the model was mtry = 2.**

This default model uses random forest and training via bootstrap. Now we run it against the test data to predict the classes thereof. We see how well it did via a confusion matrix.

**predicts <- predict(model$finalModel, testX) *# run against test data***

**confusionMatrix(predicts, testX$class)**

**## Confusion Matrix and Statistics**

**##**

**## Reference**

**## Prediction base med**

**## base 221 31**

**## med 42 232**

**##**

**## Accuracy : 0.8612**

**## 95% CI : (0.8287, 0.8896)**

**## No Information Rate : 0.5**

**## P-Value [Acc > NIR] : <2e-16**

**##**

**## Kappa : 0.7224**

**## Mcnemar's Test P-Value : 0.2418**

**##**

**## Sensitivity : 0.8403**

**## Specificity : 0.8821**

**## Pos Pred Value : 0.8770**

**## Neg Pred Value : 0.8467**

**## Prevalence : 0.5000**

**## Detection Rate : 0.4202**

**## Detection Prevalence : 0.4791**

**## Balanced Accuracy : 0.8612**

**##**

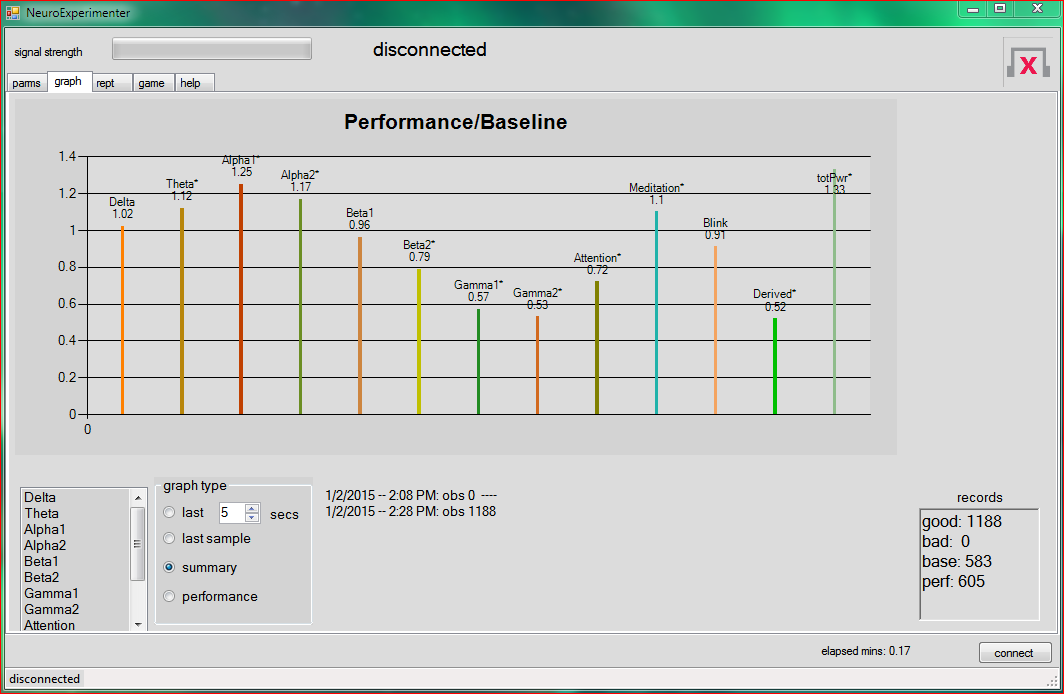
**## 'Positive' Class : base**

**##**

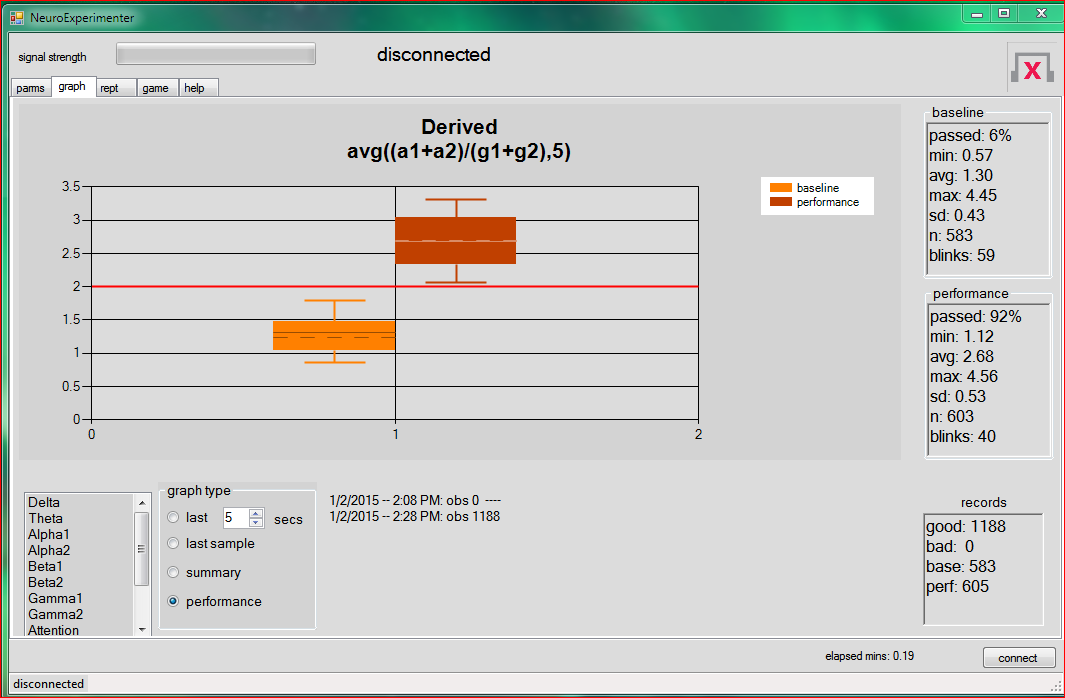
**Appendix 3**

**Relaxation Experiment**

I spent 10 minutes reading a book (baseline), and 10 minutes “relaxing”. The latter including lying down, eyes closed, mind drifting. The summary graph looked like:



The Alpha was somewhat higher, the Gamma much lower (relaxing compared to reading a book). (It is not clear to me why the blink ratio was so high). This suggested a formula of avg((a1+a2)/(g1+g2), 5). When i included a *min* of 2 and *high* of 3, the performance page produced:

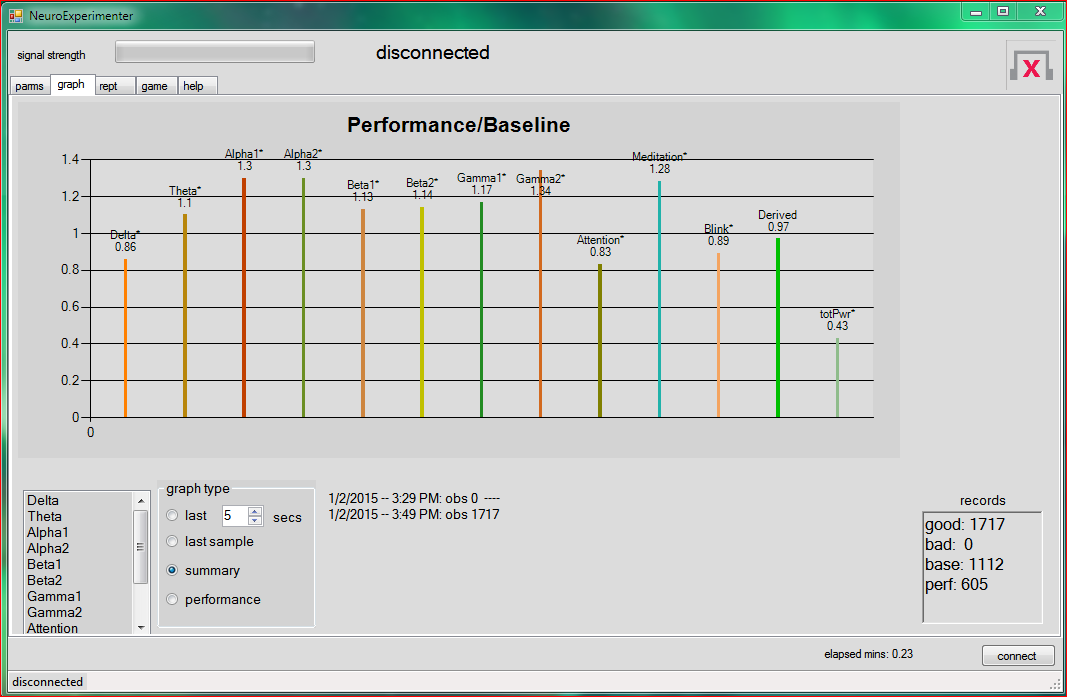


So the formula and the *min* did well with discriminating between the two states. If we take “reading a book” to be an “attention” mind state we can use these results to compare attention to relaxation. The hallmarks of attention (vs. relaxation) would then be low Alpha and high Gamma.

Note the results of my experiments on meditation and “attention” compared to the eSense output. They suggest different criteria for these states than the “[eSense](http://support.neurosky.com/kb/science/what-is-esense)” calculations. Perhaps it implies that a single algorithm cannot capture individual differences for “mind states”.

**Concentration Experiment**

I spent 10 minutes reading a book (baseline), followed by 10 minutes of solving chess problems. The summary graph:



All the power waveforms seemed to be of higher intensity (chess problem vs. reading), except for Delta, but the comparison is not as dramatic as we obtained in previous experiments. I could add up the power data, except for Delta, and then divide by Delta, but it seems too complicated a formula. I could try (a1 + a2)/d, but I would want to repeat the experiment a few times to make sure the formula was valid. For now, I conclude the system is not sensitive enough to distinguish reading a book from solving chess problems.

## Comments

These experiments in neural feedback are attempts to translate subjective mind states/feelings into objective measurements. As such, there are at least two problems:

1. The same individual might produce different brainwaves or the same brain waves at different strengths depending on her current brain chemistry. Perhaps tiredness, ingestion of alcohol or caffeine will produce different measurements from the same mind state.
2. Individuals differ in their interpretation of words like “meditation”, “relaxation”, “concentration”. The measurements produced will depend on that interpretation. For example, some experimenters belief a meditative state produces alpha waves ([http://en.wikipedia.org/wiki/Brain\_activity\_and\_meditation#Electroencephalography\_2](http://en.wikipedia.org/wiki/Brain_activity_and_meditation" \l "Electroencephalography_2)), others that it produces gamma waves ([http://en.wikipedia.org/wiki/Gamma\_wave#Relation\_to\_meditation)](http://en.wikipedia.org/wiki/Gamma_wave" \l "Relation_to_meditation)). The difference is probably explained by the subject’s own interpretation of how to meditate. Also, there might be differences in subjects’ brains (because of genetics or experience) that will lead to differences.

The experiments described in this Users’ Guide were conducted *once* by *one* person. As such they are hardly conclusive. Ensure your own results are valid by reproducing them!

Appendix 4

# Log Format and Using Other Devices

If you wish to use R to process the R log file output by NEx, you will need to understand the format. If you have another device, but wish to use NEx software, you can do so if you can create data (from your device) in the format NEx expects.

The following section documents how NEx formats an R log. However, when you create a “log file” to run in NEx (using the “replay log” function, the only lines you *need* provide are the data lines. The R “header” and the comments (lines beginning with ‘#’) are not required.

Here is a sample of an R log formatted by NEx:

# 2/19/2015 4:38 PM

time Delta Theta Alpha1 Alpha2 Beta1 Beta2 Gamma1 Gamma2 Attention Meditation Blink Derived totPwr class

# DeriveFormula: avg((g1+g2)/d, 5) #meditation: set min=1.8, high=3.5 2/19/2015 -- 4:38 PM: obs 0

# LogStarts: EMULATE 2/19/2015 -- 4:38 PM: obs 0

# StartBaseline: 2/19/2015 -- 4:38 PM: obs 0

1 2.0 173055 109172 179311 20153 323573 75893 60987 1157775 12 60 217 3.18 2099919 class\*

2 3.0 316847 62304 231182 5323 18552 95679 23349 8159 84 74 5 1.81 761395 class\*

3 4.3 115457 173712 1477600 308112 85149 312595 108369 1023026 19 26 6 2.52 3604020 class\*

4 5.3 202735 336768 152316 9862 1365552 333484 448487 1085695 59 15 63 2.84 3934899 class\*

5 6.3 1248537 1310953 106026 197633 142603 25051 75600 8709 60 10 39 2.34 3115112 class\*

…..

56 59.4 69453 1102879 97699 1290046 205948 33238 2000599 1000680 21 29 81 3.35 5800542 class\*

# EndBaseline: 2/19/2015 -- 4:39 PM: obs 56

57 60.4 1481459 38345 3901 71429 100583 304377 3 933 46 18 200 3.18 2001030 class

…….

68 71.9 374081 147349 3662 30594 41587 19472 273521 101374 5 72 66 2.78 991640 class

69 71.9 113047 15912 77479 161299 826155 430174 922 185809 NA NA NA 2.04 1810797 class

# LogEnds: 2/19/2015 -- 4:40 PM: obs 69

1. The first line is a comment (all comments start with a ‘#’ and are ignored by the R software). Comments contain the date and time of the event. Comments also contain the observation number that preceded the event (as in “obs 0”).
2. The next (second) line is the header for use in the R system.
3. The next line, # DeriveFormula: avg((g1+g2)/d, 5) #meditation: set min=1.8, high=3.5 2/19/2015 -- 4:38 PM: obs 0 is a comment showing the “derived formula” (if that is set in NEx).
4. The next lines, # LogStarts: EMULATE 2/19/2015 -- 4:38 PM: obs 0 and the one following # StartBaseline: 2/19/2015 -- 4:38 PM: obs 0 are place holders to show when the log and the baseline started.
5. A comment marking the end of the baseline is inserted, as in # EndBaseline: 2/19/2015 -- 4:39 PM: obs 56.
6. NEx ends the log with an event, as in # LogEnds: 2/19/2015 -- 4:40 PM: obs 69.
7. Data for the headset samples, as in 1 2.0 173055 109172 179311 20153 323573 75893 60987 1157775 12 60 217 3.18 2099919 class\* have this format:
   1. A sequential observation number, starting with 1 and not skipping any.
   2. A timestamp in seconds (including 10ths) when the sample was generated, beginning at the start of the session. It is probably not a good idea to have two samples with the same timestamp, but it would not be a disaster. This is used for graphing so the numbers should be reasonable.
   3. The next 11 fields are the data provided by the device. Their meaning is provided by the R header in line 2 (Theta through Blink). All are positive integers.
   4. The 14th number is the “derived” value. You should set this to NA, unless you want to calculate it for some reason. NEx does not use it when reading the log.
   5. The 15th number is the sum of the power values (i.e. Delta through Gamma2).
   6. The last entry is a class string. For samples in the baseline, the string should end in an ‘\*’. Samples in the performance section do not have the ‘\*’. This is not enforced, however. The class can be changed while NEx is connected to the headset. Press a key on the keyboard and that character becomes the new class for the next sample and all subsequent ones.
   7. Any “headset” value is allowed to be missing. Just insert an NA instead of the number. The observation number, timestamp, and class must all be present in every observation.

You must have exactly 12 fields (separated by a blank) in the data (sample) line or you will get a “format not understood” error from NEx.

The “csv” format is similar to the R format. Here is a sample:

obs, time, Delta, Theta, Alpha1, Alpha2, Beta1, Beta2, Gamma1, Gamma2, Attention, Meditation, Blink, Derived, totPwr, class

1, 1.6, 20278, 100020, 342, 724387, 1067240, 65226, 69789, 74453, 45, 83, 112, 0.01, 2121735, class\*

2, 2.6, 72687, 520, 10455, 282583, 32713, 82788, 229158, 392557, 3, 82, 122, 0.10, 1103461, class\*

3, 3.8, 16961, 1290649, 37500, 21, 87177, 89402, 67441, 297784, 9, 30, 212, 0.14, 1886935, class\*

4, 4.9, 296152, 1430223, 20086, 112101, 42137, 42573, 126662, 321011, 31, 43, 200, 0.09, 2390945, class\*

5, 5.9, 118608, 494359, 464428, 482998, 10362, 42393, 20891, 14877, 0, 25, 34, 0.53, 1648916, class\*

6, 7.0, 27429, 89223, 48929, 84728, 479254, 33740, 5131, 249586, 58, 1, 27, 0.22, 1018020, class\*

7, 8.0, 725131, 280172, 161270, 89523, 120751, 37305, 77671, 221727, 6, 53, 219, 0.31, 1713550, class\*

8, 9.1, 157882, 306227, 267192, 232528, 135786, 223103, 62174, 675331, 23, 99, 231, 0.36, 2060223, class\*

9, 9.6, 229221, 938343, 64337, 313391, 2197, 58970, 1684, 134783, 49, 58, 244, 0.19, 1742926, class\*

There are no comments (tagged via “#”) since the csv format does not allow these. The “obs” (observation) is included in the header. Only the R formatted log should be used to replay an NEx session. The csv format does not have enough information (but it can be used in a pinch if you forgot to make an R-log). The csv file can be read into Excel, R, Python, and a variety of other applications.,

There are two ways to interface your (non-MindWave) device with NEx. The first is to simply write a program that interfaces with your device and makes an R log file (as if it came from NEx). That file only needs to have lines for data (item 7 above). You can replay the log once your device has ended the session and closed the file. You start NEx with the “replay log” as the Data Source.

The second way to use NEx with your device is via a “named pipe”. You must write a program that interfaces with your device, reformats the data, and writes it to a named pipe.

In C# your code would look like:

NamedPipeServerStream server = new NamedPipeServerStream(“NexPipe”);

server.WaitForConnection();

StreamWriter writer = new StreamWriter(server);

string aLogLine = null;

while ((aLogLine == getSample()) != null)

{

writer.WriteLine(aLogLine);

writer.Flush();

}

writer.WriteLine("# LogEnds");

writer.Dispose();

Your would need to supply the function getSample() . It must wait for your device to output a sample, convert it to a string, and pass it back. It should pass NULL when there is no more data (i.e. your device becomes disconnected from your software).

The lines you put into the pipe are the data lines as documented in point 7 above. You should *not* write any other type of lines.

You should start your program and then start NEx, specifying “NexPipe” as the Data Source. You would start a session in NEx by clicking the “connect” button. This will cause NEx to process data as it is written to the pipe.

You can use all of the functions of NEx including logging and pausing. If you disconnect in NEx (by clicking “disconnect” or via session timeout) the pipe will be broken and *your* program should detect that gracefully (use try/catch). If *your* program Disposes of the pipe, NEx will disconnect normally.

Appendix 5

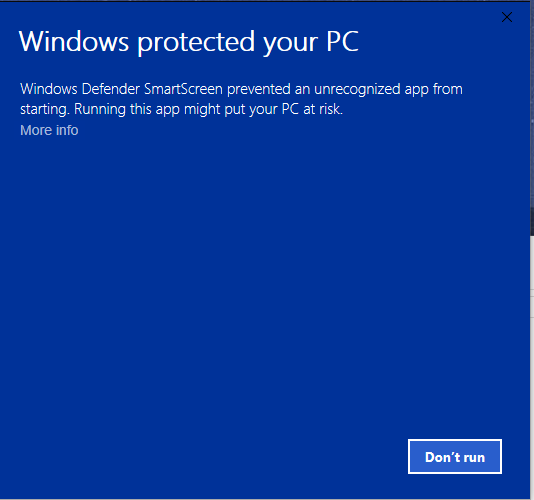
# Troubleshooting NeuroExperimenter

**Resources**

See the NeuroSky site for manuals. The MindWave 2 kit is found [here](http://mwm2.neurosky.com/" \l "Windows) .

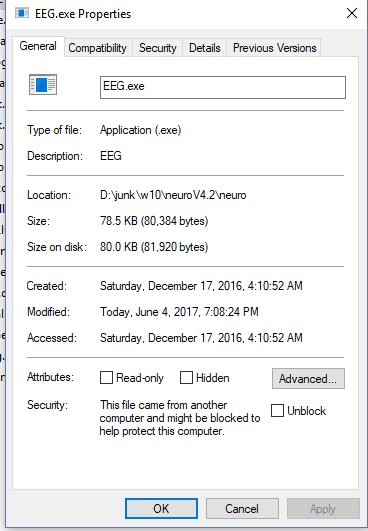
## Security Problems:

Sometimes Windows will block NEx from running. When you left-click on EEG.exe you might see:



Click ‘More Info’ and then click ‘Run anyway’.

Alternative, if you right-click on the file EEG.exe, select ‘properties’ you might see:



Click on ‘unblock’ and the program should then run.

## Non-security Problems

If you have difficulty connecting the NEx application to your headset try the following:

If you have difficulty connecting the NEx application to your headset try the following:

1. Sometimes the electrodes do not make sufficient contact. Clean them with alcohol. If that does not work try the following: take a dab of skin cream (any brand) and mix a little salt into it. Put the cream on your ear and forehead where the contacts will touch. You can buy “electrode gel” too, but that should not be necessary.
2. See if you can run other applications (that come with the headset). Follow the instructions that came with the application. If it works with that, it should work with NEx.
3. Use the “test headset” tab on NEx. You need to get good output to assure that NEx will

function properly. You should only have to run this once (when you install NEx), unless

NEx has problems connecting with the headset.

### Bluetooth Settings

If you still have difficulty because you cannot pair the headset with the computer, it may be a problem with your Bluetooth settings. Sometimes Windows patches will reset these and you have to set them again. Note: Bluetooth applies to MindWave mobile (black headset).

You must configure bluetooth correctly. Below, I show how this would look (more or less) when NEx is properly configured. Your goal is to achieve this.

In Windows 10 here is the procedure (in Windows 7 and 8 it may be a bit different):.

For Mindwave Mobile, with the 3 position switch, hold the switch up until the red LED shows.

That clears any previous pairing setting in the headset. For the two position switch, just turn it

on.

Next, right click on the BlueTooth icon in the taskbar. Click on the menu item: *Show Bluetooth*

*devices.* If there is no icon you can go to Settings/Devices/Bluetooth . The Bluetooth

window (Windows 10) looks like:



Scroll down to look at all the devices and see if “Mindwave Mobile” is there (under “Other

Devices”).

If you see the MindWave headset, delete it so we can start afresh: click on the device and from

the popup button, click on “Remove device”.

Now, click on “Add Bluetooth or other devices” (at the top of the window -- scroll up) Make sure

the slider “Bluetooth” is “on”. Click on “Bluetooth” in the “Add a device” window that pops up.

From the list, click on “Bluetooth”.

Turn on the headset. Click on “Unknown device”, and the device should be added to the list.

When the device shows in the list, click on it and eventually it should say “Paired”. If it asked

you for a pin, try “0000”. Then click the “Done” button. That should work to pair the Mindwave

Mobile Version 1 and version 2.

Returning to the “Bluetooth & other devices” window, you should see the device in the “Other

devices list:

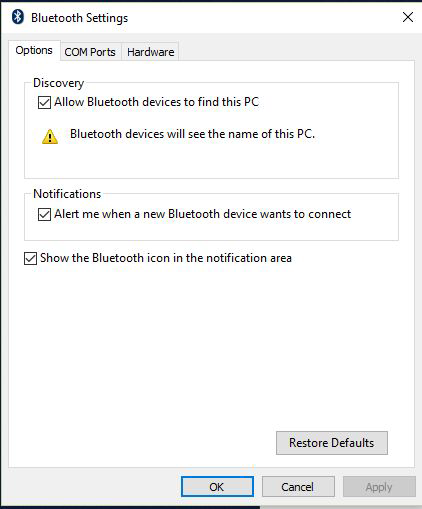


After a few seconds, the headset (version 1) should have a slow, single blink and show up as

paired. Version 2 output is just a solid blue light and does not indicate whether you are paired or

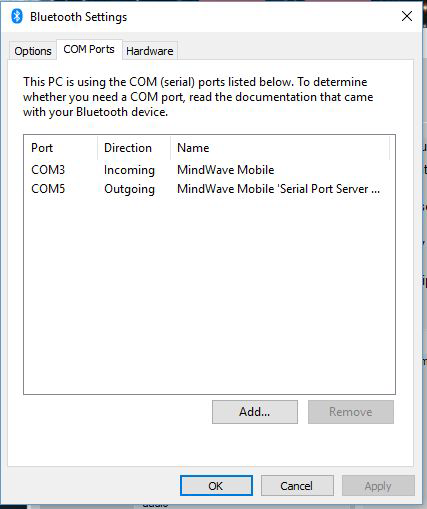
trying to pair.

Now, click on “More Bluetooth options”: You should see the following window:



Make sure you have Discovery checked, along with the other options.

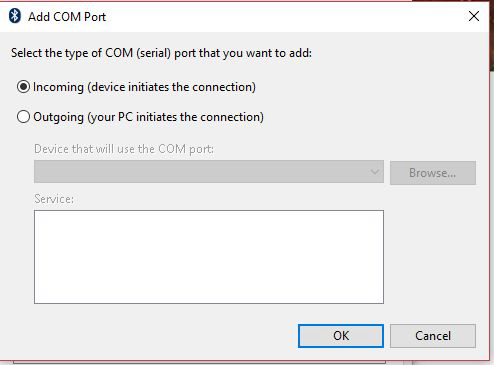
Click on COM ports to see:



The actual COM numbers will differ and you may see none at all. The pairing should have set

these COM ports. If you don’t see them, you should try to set them up (this is unusual: the

pairing should have set them up). If necessary, click on the *add button* to add an Incoming COM port. Then you will see:



Check the Incoming button and click on OK to add an incoming device. You should not have to

add an Outgoing port: the headset will do that. Now try to pair the headset by following the

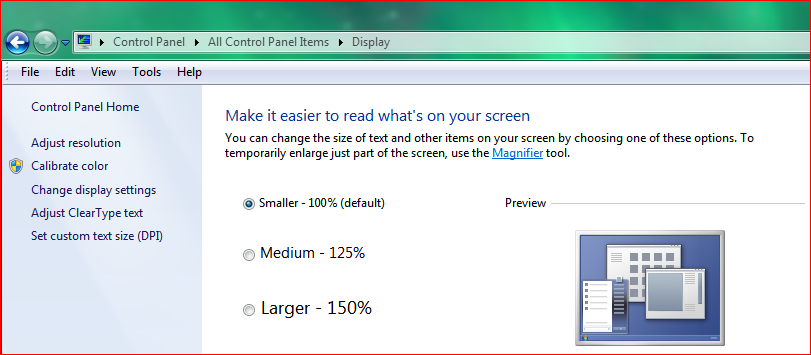
instructions in the test-headset procedure (above).

## Known Bugs in NeuroExperimenter

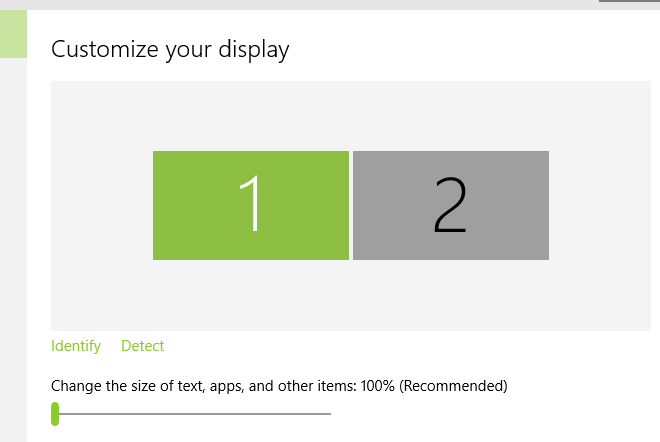
1. The NEx window does not seem to fit the display.

The resolution of your display must be at least 1024 x 768. Furthermore, you must have the text size set to “Smaller” via the control panel. Otherwise, the windows will not display correctly.

Here is the control under Windows 7.



Under Windows 10 it looks like:



and should be set to 100%. If you right-click on the desktop you will see these settings.

Note: as of version 3.28, provided your display has a high enough resolution (larger than 1250x900 for 1.25%), this (might) no longer be a problem under Windows 10.

2. You get “Could not connect ….” in a pop-up window.

First of all, be sure the headset has been paired and that it is turned on. You might also try other

applications that came with the headset. See step 1 on page 45 to set up the device in

Windows. If it is already there, delete it and set it up again (following the instructions above).

You *must* have the device set up and paired in Windows. Furthermore, the blue light must be

flashing once (slowly, MIndwave Mobile 1, not MWM 2). The blue light has a double flash when

it is trying to pair with Windows. Even though Windows says it is paired, it is not discovered until the blue light has a single, slow flash. The Mindwave Mobile 2 does not indicate if it is paired,

not paired, or trying to pair.

**How to Report Bugs**

1. Email me the information at [fredm73@gmail.com](mailto:frem73@gmail.com)
2. Tell me the version/model/color of the Neurosky hardware/headset (e.g. Mindwave Mobile, black).
3. Tell me the version of Windows you are running: Window 7, 8, what? If NEx runs at all, you can go to the help page and cut/paste all version information.
4. Include the following log files, if present, (in the folder that contains the EEG.exe): log.txt and/or fredm.elog.txt
5. Include any screenshots you think would be helpful. If you get a pop-up window that shows an error you can copy the text: put your cursor in the window, press CNTL-C, then paste it into your email via CNTL-V. You can use the [snipping tool](http://windows.microsoft.com/en-us/windows7/products/features/snipping-tool) to capture the window too, as well as CNTL-PrtScn.
6. Include the output from the TestHeadSet program. You can copy it to the clipboard and then paste it into the email.

Appendix 6: Change Log

Version 3.21: I changed the timestamp (on the log and the graph) to use that given by the NeuroSky hardware (for each sample). Before, I was tagging the sample internally in my app with the system time of the sample arrival into my app. Because the Windows OS can delay my app for various reasons (e.g. garbage collection), and the NeuroSky hardware can buffer the data before it is requested by the app., the new logic should result in slightly higher accuracy.

Version 3.22: I attempted to ameliorate the bug in Windows 8/10 (or in NeuroSky code).

The proper place for this fix is in NeuroSky code, not in NEx. However, they believe it is a bug in Windows 8/10 and cannot seem to find a work around. Hence I am sorry to impose this kludge on my users.

The problem is that the ConnectScan call throws a (synchronous) Exception:

NeuroSky.ThinkGear.Connector.FindAvailablePorts()

NeuroSky.ThinkGear.Connector.ConnectScan(String initialPort)

EEGlibrary.HeadSet.startup()

One of my users discovered that if the COM port passed in the initialPort is the one actually holding the headset, then ThinkGear.dll will go ahead and connect, despite having thrown the exception.

I changed the TestHeadset program to change the port number in the xconfig.xml file to the one holding the headset. Of course, if the connect is not successful in the TestHeadset, the file is not changed and will have to be manually changed by the user. I changed the NEx program to catch the Exception thrown by NeuroSky code by ConnectScan. I was very reluctant to do this: the Exception should be trapped in NeuroSky code, but I could not arrange for that. If NeuroSky’s ThinkGear.dll eventually does connect , the Exception is suppressed and NEx should continue normally. If we are unable to connect before a timeout (about 15 seconds) the Exception is reported in a pop-up window, along with a stack trace. Then the user should take the action described in Appendix 5 (i.e. manually change the port in the xConfig file). It NEx still fails, there is no solution.

Note: in August 2015 I installed Windows 10. It seems that the bug persists under Windows 10.

ThinkGear.dll is marked as version: 2.8.4892.19806. Perhaps a new version will appear and fix the problem. Furthermore, I found that Windows 10 on my laptop, with built-in Bluetooth failed to connect. I installed a USB Bluetooth device (<http://plugable.com/products/usb-bt4le>) and it worked fine. However, you must uninstall the Microsoft bluetooth drivers (see the FAQ at the plugable site). Note that the latest (huge) update of Windows 10 (on 12/02/2015) reinstalled the MS drivers and once again, my laptop could not find the headset. I deleted the MS drivers as well as the Plugable ones, then plugged in the Plugable device and things worked fine, again.

Versions 3.24, 3.25...: A sound and message is output if, after a good connection, the headset signal is lost. This can happen if the headset goes out of range, its battery becomes weak, etc. I’m still not entirely satisfied with the error handling and continue to tweak it.

Version 3.28: changed Autoscalemode to None so that user can adjust Display Settings for larger fonts. This results in some fuzziness of the text, but at least the window contents are not clipped.

Version 3.30: added a checkbox to create a “csv” log.

Version 4.1: When you press a key when the headset is connected, NEx sets the ‘class’ of the next sample and all subsequent ones (in the log) to that character. You can tag samples this way when an external event occurs. You can write a program/spreadsheet that analyzes the log based on class values. You should probably normalize the data before processing it.

Version 4.6: Added the ability to use a wave file instead of a tone for audio feedback. Saved

more parameters for the next startup. Fixed some bugs.

Version 4.8: Added two lines to the performance box. The first is the longest unbroken interval

where all samples passed the grader, the second is the time stamp when that interval ended.

Added a parameter for the minimum signal strength for a sample to be included in the interval.

Version 5.2: I converted NEx to use the latest SDK from NeuroSky. I added another option for

audio “pink” which gives “ pink noise ” for output.

Version 5.7: I completely redid the interface after NeuroSky discontinued the old one. I was

unsuccessful in getting version 3.2 of the SDK (.Net dll) to work. I am now using the Serial

Stream Protocol . However, this does not support “blink” data and it does not seem to work with

the white headset. It seems to work fine with MWM-1 and MWM-2 (black headsets).